APPENDIX

Serial No.: _____

Title:
Open Architecture For A Voice User Interface

Inventors:
Eugene Ponomarenko
Mitsuru Oshima
Daniel Burkes
Min _____
Daniel Lange
Marianna Tessel

Filing Date: December 8, 2000

Attorney Docket Number: M-9906 US

```
/*
 * AcceptCallMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:21:05 $
 */
#ifndef ACCEPTCALLMSG_H
#define ACCEPTCALLMSG_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
#include "CDRInfo.h"
class PropertyList;
class __declspec(dllexport) AcceptCallMessage : public OneCallTokenMessageBase
{
        PropertyList *cProperties;
        bool cOwned;
        CDRInfo cCDRInfo;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(AcceptCallMessage, OneCallTokenMessageBase);
        inline AcceptCallMessage(CallToken& token, PropertyList *pProps, CDRInfo& cdr);
        virtual ~AcceptCallMessage(void);
        inline PropertyList *GetProperties(void);
        inline CDRInfo& GetCDRInfo(void);
};
inline
AcceptCallMessage::AcceptCallMessage(CallToken& token, PropertyList *pProps, CDRInfo& cdr)
        : OneCallTokenMessageBase(M_CMD_ACCEPTCALL, token), cProperties(pProps), cCDRInfo(cdr)
{
        cOwned = false;
}
inline PropertyList *
AcceptCallMessage::GetProperties(void)
{
        return cProperties;
}
inline CDRInfo&
AcceptCallMessage::GetCDRInfo(void)
{
        return cCDRInfo;
}
#endif
```

```
/*
 * AcuvoiceTTSChannel.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.6 $
 * $Date: 2000/12/07 03:34:27 $
 */
#ifdef  AVCORE
#include "AVCORE.h"
#else
#include "av2001.h"
#endif
#include "MTSLib/TTSChannel.h"
#include "VPPUtils/Synchronized.h"
#ifndef  AVCORE
typedef  struct SSB          *SSB_PTR;
#endif
class ServiceProvider;
class __declspec(dllexport) AcuvoiceTTSChannel : public TTSChannel
{
public:
        AcuvoiceTTSChannel::AcuvoiceTTSChannel(ServiceProvider *pServiceProvider, const char *cLogType, int
id);
        virtual bool Shutdown(void);
        virtual ~AcuvoiceTTSChannel(void);
protected:
        virtual bool StartTTS(void);        // Get the tts engine started on m_pString data
        virtual bool NextTTS(void);                // Get the tts engine started on next chunk of text
        virtual void StopTTS(void);                // Halt tts processing of text, get ready for new string
        virtual TTSChannel::TTSSynthStatus SynthesisStatus(void);
        virtual int GetNextBuffer(char *buffer, unsigned long m_bufferSize);
private:
        // Acuvoice buffer max buffer size defined by acuvoice dll
        enum {
                AccuvoiceBufSize = 500 * 1024,
        // MarginDataSize is the maximum amount of audio data left,
        // from a previous tts engine request, before we
        // make another tts request.  This allows us to use this
        // data while the next chunk of text is being processed.
        // Should be at least twice the request buffer size from the media channel
                MarginDataSize  = 2 * 8000        // 8000 bytes gives one second of audio
        };
        typedef enum
        {
                TTSRequest_Start,
                TTSRequest_Next,
                TTSRequest_Stop,
                TTSRequest_Exit
        } TTSRequest;
        void StartWorkerThread(void);      // Starts _TTSChannelThread for channel
        bool SendTtsRequest(TTSRequest request); // Issue request to channel thread
```

```
    static void _TTSChannelThread(void *);      // Thread for each channel that calls TTSProcess
    void TTSProcess(void);    // Wait for and handles channel thread requests
    void ProcessTTSRequest_Start(void);
    void ProcessTTSRequest_Next(void);
    void ProcessTTSRequest_Stop(void);
    void CheckBuffer(void);             // Check for need to switch
    HANDLE              m_EventSignalHandle;
    unsigned long   m_ThreadHandle;
    Monitor         m_Monitor;
    Monitor         m_WorkerMonitor;
    TTSRequest ttsRequest;
    char    *audioBuffer;     // Buffer where engine returns output.
    char    *audioBufInPtr;             // Buffer where engine returns output
    char    *audioBufOutPtr;// Pointer to data for output of GetNextBuffer
    int             audioBufOutRemaining;
    bool    synthisizing;

    // Variables set and used by worker thread
    bool    startedSynth;
    SSB_PTR m_SSBPtr;       // Acuvoice Synthesis Status Block
    int             dataAvail;          // Indicates TTS engine has more data to return
    DWORD           synSize; // Size of the last sample block received from engine.
};
```

```
/*
 * AcuvoiceTTSSP.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:34:27 $
 */
#ifndef ACUVOICE_H
#define ACUVOICE_H
#include "MTSLib/TTSServiceProvider.h"
class __declspec(dllexport) AcuvoiceTTSSP : public TTSServiceProvider
{
private:
        static const char const *cLogType;
public:
        inline AcuvoiceTTSSP(MTSServer *pServer);
        ~AcuvoiceTTSSP(void);
        virtual bool Setup(void);
        virtual bool Teardown(void);
};
inline
AcuvoiceTTSSP::AcuvoiceTTSSP(MTSServer *pServer)
        : TTSServiceProvider(pServer, cLogType)
{
}
#endif
```

Confidential - Do Not Distribute

```
/*
 * AddPhraseToDynamicGrammarMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:06 $
 */
#ifndef ADDPHRASETODYNAMICGRAMMARMSG_H
#define ADDPHRASETODYNAMICGRAMMARMSG_H
#include "DynamicGrammarPhraseMsg.h"
class __declspec(dllexport) AddPhraseToDynamicGrammarMessage : public DynamicGrammarPhraseMessage
{
        gm::string cPhraseText;
        gm::string cPhraseNL;
        unsigned long cProbability;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(AddPhraseToDynamicGrammarMessage,
DynamicGrammarPhraseMessage);
        inline AddPhraseToDynamicGrammarMessage(CallToken& token, const gm::string& db_key,
                const gm::string &phrase_id, const gm::string& phrase_text, const gm::string& phrase_nl,
                unsigned long probability);
        inline gm::string& GetPhraseText(void);
        inline gm::string& GetPhraseNL(void);
        inline unsigned long GetProbability(void);
};
inline
AddPhraseToDynamicGrammarMessage::AddPhraseToDynamicGrammarMessage(CallToken& token,
        const gm::string& db_key, const gm::string &phrase_id, const gm::string& phrase_text,
        const gm::string& phrase_nl, unsigned long probability)
        : DynamicGrammarPhraseMessage(M_CMD_ADDPHRASETODYNAMICGRAMMAR, token, db_key,
phrase_id),
        cPhraseText(phrase_text), cPhraseNL(phrase_nl), cProbability(probability)
{
}
inline gm::string&
AddPhraseToDynamicGrammarMessage::GetPhraseText(void)
{
        return cPhraseText;
}
inline gm::string&
AddPhraseToDynamicGrammarMessage::GetPhraseNL(void)
{
        return cPhraseNL;
}
inline unsigned long
AddPhraseToDynamicGrammarMessage::GetProbability(void)
{
        return cProbability;
```

```
}
#endif
```

```
/*
 * AddPhraseToDynamicGrammarResultNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:06 $
 */
#ifndef ADDPHRASETODYNAMICGRAMMARRESULTNTFY_H
#define ADDPHRASETODYNAMICGRAMMARRESULTNTFY_H
#include "DynamicGrammarPhraseResultNtfy.h"
class __declspec(dllexport) AddPhraseToDynamicGrammarResultNotification : public
DynamicGrammarPhraseResultNotification
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(AddPhraseToDynamicGrammarResultNotification,
                DynamicGrammarPhraseResultNotification);
        inline AddPhraseToDynamicGrammarResultNotification(CallToken& ct, const gm::string& db_key, Result
result,
                const gm::string& phrase_id);
};
inline
AddPhraseToDynamicGrammarResultNotification::AddPhraseToDynamicGrammarResultNotification(CallToken&
callToken,
        const gm::string& db_key, DynamicGrammarResultNotification::Result result, const gm::string& phrase_id)
        : DynamicGrammarPhraseResultNotification(M_NTFY_ADDPHRASETODYNAMICGRAMMARRESULT,
callToken, db_key, result, phrase_id)
{
}
#endif
```

```
/*
 * Address.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:21:06 $
 */
#ifndef ADDRESS_H
#define ADDRESS_H
#include "VPPUtils/GMString"
class VPPMessage;
class __declspec(dllexport) Address
{
        gm::string cAddress;
public:
        inline Address(void);
        inline Address(const gm::string& addr);
        inline void Setup(const gm::string& addr);
        inline const gm::string& GetString(void);
        void Serialize(VPPMessage *);
        void Deserialize(VPPMessage *);
};
inline
Address::Address(void)
{
}
inline void
Address::Setup(const gm::string& addr)
{
        cAddress = addr.c_str();
}
inline
Address::Address(const gm::string& addr)
{
        Setup(addr);
}
inline const gm::string&
Address::GetString(void)
{
        return cAddress;
}
#endif
```

```
/*
 * AppendCompositePromptDTMFMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:06 $
 */
#ifndef APPENDCOMPOSITEPROMPTDTMFMSG_H
#define APPENDCOMPOSITEPROMPTDTMFMSG_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
#include "VPPUtils/GMString"
class __declspec(dllexport) AppendCompositePromptDTMFMessage :
        public OneCallTokenMessageBase
{
   gm::string cDTMFString;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(AppendCompositePromptDTMFMessage,
                OneCallTokenMessageBase);
        inline AppendCompositePromptDTMFMessage(CallToken& token,
                const gm::string& dtmfString);
        inline const gm::string& GetDTMFString(void);
};
inline
AppendCompositePromptDTMFMessage::AppendCompositePromptDTMFMessage(
        CallToken& token, const gm::string& dtmfString)
        : OneCallTokenMessageBase(M_CMD_APPENDCOMPOSITEPROMPTDTMF, token)
{
        cDTMFString = dtmfString;
}
inline const gm::string&
AppendCompositePromptDTMFMessage::GetDTMFString(void)
{
        return cDTMFString;
}
#endif
```

```
/*
 * AppendCompositePromptPromptMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:21:06 $
 */
#ifndef APPENDCOMPOSITEPROMPTPROMPTMSG_H
#define APPENDCOMPOSITEPROMPTPROMPTMSG_H
#include "PlayMsg.h"
class __declspec(dllexport) AppendCompositePromptPromptMessage :
        public PlayMessage
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(AppendCompositePromptPromptMessage,
                PlayMessage);
        inline AppendCompositePromptPromptMessage(CallToken& token,
                const gm::string& promptname, const gm::string& productname,
                GMLANGID lang, const gm::string& tts_alternate);
        inline AppendCompositePromptPromptMessage(CallToken& token,
                const gm::string& path, const gm::string& tts_alternate, bool bUseSearchPath=false);
};
inline
AppendCompositePromptPromptMessage::AppendCompositePromptPromptMessage(
        CallToken& token, const gm::string& promptname,
        const gm::string& productname, GMLANGID lang, const gm::string& tts_alternate)
        : PlayMessage(token, promptname, productname, lang, tts_alternate)
{
        SetCommandID(M_CMD_APPENDCOMPOSITEPROMPTPROMPT);
}
inline
AppendCompositePromptPromptMessage::AppendCompositePromptPromptMessage(CallToken& token,
        const gm::string& path,     const gm::string& tts_alternate, bool bUseSearchPath)
        : PlayMessage(token, path, tts_alternate, bUseSearchPath)
{
        SetCommandID(M_CMD_APPENDCOMPOSITEPROMPTPROMPT);
}
#endif
```

```
/*
 * AppendCompositePromptSilenceMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:07 $
 */
#ifndef APPENDCOMPOSITEPROMPTSILENCEMSG_H
#define APPENDCOMPOSITEPROMPTSILENCEMSG_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) AppendCompositePromptSilenceMessage :
        public OneCallTokenMessageBase
{
        unsigned long cDuration;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(AppendCompositePromptSilenceMessage,
                OneCallTokenMessageBase);
        inline AppendCompositePromptSilenceMessage(CallToken& token,
                unsigned long duration);
        inline unsigned long GetDuration(void);
};
inline
AppendCompositePromptSilenceMessage::AppendCompositePromptSilenceMessage(
        CallToken& token, unsigned long duration)
        : OneCallTokenMessageBase(M_CMD_APPENDCOMPOSITEPROMPTSILENCE, token)
{
        cDuration = duration;
}
inline unsigned long
AppendCompositePromptSilenceMessage::GetDuration(void)
{
        return cDuration;
}
#endif
```

```
/*
 * AppendCompositePromptTTSMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:07 $
 */
#ifndef APPENDCOMPOSITEPROMPTTTSMSG_H
#define APPENDCOMPOSITEPROMPTTTSMSG_H
#include "TTSMsg.h"
class __declspec(dllexport) AppendCompositePromptTTSMessage :
        public TTSMessage
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(AppendCompositePromptTTSMessage,
                TTSMessage);
        inline AppendCompositePromptTTSMessage(CallToken& token,
                const gm::string& text, const gm::string& productname, GMLANGID lang,
                unsigned short Speed, unsigned short Pitch);
};
inline
AppendCompositePromptTTSMessage::AppendCompositePromptTTSMessage(
        CallToken& token, const gm::string& text, const gm::string& productname,
        GMLANGID lang, unsigned short Speed, unsigned short Pitch)
        : TTSMessage(token, text, productname, lang, Speed, Pitch)
{
        SetCommandID(M_CMD_APPENDCOMPOSITEPROMPTTTS);
}
#endif
```

```
/*
 * BeginSpeechNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:07 $
 */
#ifndef BEGINSPEECHNTFY_H
#define BEGINSPEECHNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) BeginSpeechNotification : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(BeginSpeechNotification,
                OneCallTokenMessageBase);
        inline BeginSpeechNotification(CallToken& token);
};
inline
BeginSpeechNotification::BeginSpeechNotification(CallToken& token)
        : OneCallTokenMessageBase(M_NTFY_BEGINSPEECH, token)
{
}
#endif
```

```
/*
 * CallAcceptedNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:09 $
 */
#ifndef CALLACCEPTEDNTFY_H
#define CALLACCEPTEDNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) CallAcceptedNotification : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(CallAcceptedNotification,
                OneCallTokenMessageBase);
        inline CallAcceptedNotification(CallToken& token);
};
inline
CallAcceptedNotification::CallAcceptedNotification(CallToken& token)
        : OneCallTokenMessageBase(M_NTFY_CALLACCEPTED, token)
{
}
#endif
```

```
/*
 * CallConnectedNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:21:09 $
 */
#ifndef CALLCONNECTEDNTFY_H
#define CALLCONNECTEDNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
#include "RequestToken.h"
class __declspec(dllexport) CallConnectedNotification : public OneCallTokenMessageBase
{
        RequestToken cRequest;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(CallConnectedNotification,
                OneCallTokenMessageBase);
        inline CallConnectedNotification(RequestToken& request, CallToken& ct);
        inline RequestToken& GetRequestToken(void);
};
inline
CallConnectedNotification::CallConnectedNotification(RequestToken& request,
        CallToken& ct) : OneCallTokenMessageBase(M_NTFY_CALLCONNECTED, ct)
{
        cRequest = request;
}
inline RequestToken&
CallConnectedNotification::GetRequestToken(void)
{
        return cRequest;
}
#endif
```

```
/*
 * CallDroppedNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:09 $
 */
#ifndef CALLDROPPEDNTFY_H
#define CALLDROPPEDNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) CallDroppedNotification : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(CallDroppedNotification,
                OneCallTokenMessageBase);
        inline CallDroppedNotification(CallToken& token);
};
inline
CallDroppedNotification::CallDroppedNotification(CallToken& token)
        : OneCallTokenMessageBase(M_NTFY_CALLDROPPED, token)
{
}
#endif
```

```
/*
 * CallPlacedOnHoldNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:09 $
 */
#ifndef CALLPACEDONHOLDNTFY_H
#define CALLPLACEDONHOLDNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) CallPlacedOnHoldNotification : public OneCallTokenMessageBase
{
        public:
                DECLARE_CMSMESSAGE_SUBCLASS(CallPlacedOnHoldNotification,
                        OneCallTokenMessageBase);
                inline CallPlacedOnHoldNotification(CallToken& token);
};
inline
CallPlacedOnHoldNotification::CallPlacedOnHoldNotification(CallToken& token)
        : OneCallTokenMessageBase(M_NTFY_CALLPLACEDONHOLD, token)
{
}
#endif
```

```
/*
 * CallProgressNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.8 $
 * $Date: 2000/12/07 03:21:10 $
 */
#ifndef CALLPROGRESSNTFY_H
#define CALLPROGRESSNTFY_H
#include "VPPDefs.h"
#include "RequestTokenMsgBase.h"
class __declspec(dllexport) CallProgressNotification : public RequestTokenMessageBase
{
public:
        typedef enum
        {
                //CPN_Queued,
                CPN_Proceeding,
    CPN_Alerting,
                CPN_Busy,
                CPN_NoRingback,
                CPN_NoDialtone,
                CPN_NoAnswer,
                CPN_UserBusy,
                CPN_NetworkBusy,
                CPN_InvalidNumber,
                CPN_CallRejected,
                CPN_NetworkError,
                CPN_Error
        } CallProgressT;
private:
        CallProgressT cNotification;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);

public:
        DECLARE_CMSMESSAGE_SUBCLASS(CallProgressNotification, RequestTokenMessageBase);
        inline CallProgressNotification(RequestToken& request,
                CallProgressT notification);
        inline CallProgressT GetNotification(void);
};
inline
CallProgressNotification::CallProgressNotification(RequestToken& request,
        CallProgressT notification) : RequestTokenMessageBase(M_NTFY_CALLPROGRESS, request)
{
        cNotification = notification;
}
inline CallProgressNotification::CallProgressT
CallProgressNotification::GetNotification(void)
{
```

```
        return cNotification;
}
#endif
```

Document provided by IPFS

```
/*
 * CallRejectedNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:10 $
 */
#ifndef CALLREJECTEDNTFY_H
#define CALLREJECTEDNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) CallRejectedNotification : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(CallRejectedNotification,
                OneCallTokenMessageBase);
        inline CallRejectedNotification(CallToken& token);
};
inline
CallRejectedNotification::CallRejectedNotification(CallToken& token)
        : OneCallTokenMessageBase(M_NTFY_CALLREJECTED, token)
{
}
#endif
```

```
/*
 * CallsConnectedNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:11 $
 */
#ifndef CALLSCONNECTEDNTFY_H
#define CALLSCONNECTEDNTFY_H
#include "TwoCallTokensMsgBase.h"
class __declspec(dllexport) CallsConnectedNotification : public TwoCallTokensMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(CallsConnectedNotification,
                TwoCallTokensMessageBase);
        inline CallsConnectedNotification(CallToken& call1,
                CallToken& call2);
};
inline
CallsConnectedNotification::CallsConnectedNotification(
        CallToken& call1, CallToken& call2) : TwoCallTokensMessageBase(
        M_NTFY_CALLSCONNECTED, call1, call2)
{
}
#endif
```

```
/*
 * CallsDisconnectedNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:11 $
 */
#ifndef CALLSDISCONNECTEDNTFY_H
#define CALLSDISCONNECTEDNTFY_H
#include "TwoCallTokensMsgBase.h"
class __declspec(dllexport) CallsDisconnectedNotification : public TwoCallTokensMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(CallsDisconnectedNotification,
                TwoCallTokensMessageBase);
        inline CallsDisconnectedNotification(CallToken& call1,
                CallToken& call2);
};
inline
CallsDisconnectedNotification::CallsDisconnectedNotification(
        CallToken& call1, CallToken& call2) : TwoCallTokensMessageBase(
        M_NTFY_CALLSDISCONNECTED, call1, call2)
{
}
#endif
```

```
/*
 * CallToken.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.8 $
 * $Date: 2000/12/07 03:21:10 $
 */
#ifndef CALLTOKEN_H
#define CALLTOKEN_H
#if 0
#include "VMachineID.h"
#endif
class VPPMessage;
class __declspec(dllexport) CallToken
{
        unsigned short cSpan;
        unsigned short cChannel;
        unsigned long cSequenceNumber;
        unsigned long cTSPPrivate;
#if 0
        VirtualMachineID cMTSVirtualMachineID;
#endif
public:
        inline CallToken(void);
        inline CallToken(unsigned short Span, unsigned short Channel,
                unsigned long SequenceNumber, unsigned long TSPPrivate = 0);
        inline void SetSpan(unsigned short Span);
        inline unsigned short GetSpan(void);

        inline void SetChannel(unsigned short Channel);
        inline unsigned short GetChannel(void);
        inline void SetSequenceNumber(unsigned long SequenceNumber);
        inline unsigned long GetSequenceNumber(void);
        inline void SetTSPPrivate(unsigned long TSPPrivate);
        inline unsigned long GetTSPPrivate(void);
#if 0
        inline void SetMTSVirtualMachineID(VirtualMachineID& MTSVirtualMachineID);
        inline VirtualMachineID& GetMTSVirtualMachineID(void);
#endif
        inline void Invalidate(void);
        inline bool IsValid(void);
        inline bool operator==(const CallToken& other);
        inline bool operator!=(const CallToken& other);

        void Serialize(VPPMessage *);
        void Deserialize(VPPMessage *);
        enum
        {
                INVALID_SPAN = (unsigned short)-1,
                INVALID_CHANNEL = (unsigned short)-1,
                INVALID_SEQUENCENUMBER = (unsigned long)-1
```

```
        };
};
inline void
CallToken::Invalidate(void)
{
        cSpan = INVALID_SPAN;
        cChannel = INVALID_CHANNEL;
        cSequenceNumber = INVALID_SEQUENCENUMBER;
}
inline
CallToken::CallToken(void)
{
        Invalidate();
        cTSPPrivate = 0;
}
inline
CallToken::CallToken(unsigned short Span, unsigned short Channel,
        unsigned long SequenceNumber, unsigned long TSPPrivate)
{
        cSpan = Span;
        cChannel = Channel;
        cSequenceNumber = SequenceNumber;
        cTSPPrivate = TSPPrivate;
}
inline void
CallToken::SetSpan(unsigned short Span)
{
        cSpan = Span;
}
inline unsigned short
CallToken::GetSpan(void)
{
        return cSpan;
}
inline void
CallToken::SetChannel(unsigned short Channel)
{
        cChannel = Channel;
}
inline unsigned short
CallToken::GetChannel(void)
{
        return cChannel;
}
inline void
CallToken::SetSequenceNumber(unsigned long SequenceNumber)
{
        cSequenceNumber = SequenceNumber;
}
inline unsigned long
CallToken::GetSequenceNumber(void)
{
        return cSequenceNumber;
}
inline void
CallToken::SetTSPPrivate(unsigned long TSPPrivate)
```

```
{
        cTSPPrivate = TSPPrivate;
}
inline unsigned long
CallToken::GetTSPPrivate(void)
{
        return cTSPPrivate;
}
#if 0
inline void
CallToken::SetMTSVirtualMachineID(VirtualMachineID& MTSVirtualMachineID)
{
        cMTSVirtualMachineID = MTSVirtualMachineID;
}
inline VirtualMachineID&
CallToken::GetMTSVirtualMachineID(void)
{
        return cMTSVirtualMachineID;
}
#endif
inline bool
CallToken::IsValid(void)
{
        return (bool)((cSpan != INVALID_SPAN) &&
                (cChannel != INVALID_CHANNEL) &&
                (cSequenceNumber != INVALID_SEQUENCENUMBER));
#if 0
                && (cMTSVirtualMachineID.IsValid())
                );
#endif
}
inline bool
CallToken::operator==(const CallToken& other)
{
        return (bool)((cSpan == other.cSpan) &&
                (cChannel == other.cChannel) &&
                (cSequenceNumber == other.cSequenceNumber) &&
                (cTSPPrivate == other.cTSPPrivate));
#if 0
                && (cMTSVirtualMachineID == other.cMTSVirtualMachineID)
                );
#endif
}
inline bool
CallToken::operator!=(const CallToken& other)
{
        return !operator==(other);
}
#endif
```

```
/*
 * CallTokenMgr.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:20:05 $
 */
#ifndef _CALLTOKEN_MGR_H
#define _CALLTOKEN_MGR_H
#include "invui_c.h"
#include "VPPUtils/Hashtable.h"
#include "VPPUtils/Queue.h"
#include "VPPUtils/Monitor.h"
#include "VPPUtils/Synchronized.h"
class CallToken;
class CallTokenMgr {
   static Hashtable<CORBA::LongLong, CallToken>* id_to_ct;
   static Hashtable<CallToken, CORBA::LongLong>* ct_to_id;
   static Queue<CORBA::LongLong> tokens;
public:
   static CallToken GetCallToken(CORBA::LongLong id);
   static CORBA::LongLong GetCallID(CallToken& token);
   static void Register(CORBA::LongLong id, CallToken& token);
};
#endif
```

```
/*
 * CancelRequestMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:21:11 $
 */
#ifndef CANCELREQUEST_H
#define CANCELREQUEST_H
#include "VPPDefs.h"
#include "RequestTokenMsgBase.h"
class __declspec(dllexport) CancelRequestMessage : public RequestTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(CancelRequestMessage, RequestTokenMessageBase);
        inline CancelRequestMessage(RequestToken& request);
};
inline
CancelRequestMessage::CancelRequestMessage(RequestToken& request)
        : RequestTokenMessageBase(M_CMD_CANCELREQUEST, request)
{
}
#endif
```

```
/*
 * CDRInfo.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:21:08 $
 */
#ifndef CDRINFO_H
#define CDRINFO_H
#include <VPPUtils/GMString>
class VPPMessage;
class __declspec(dllexport) CDRInfo
{
        gm::string cServiceName;
        gm::string cBillingAccount;
        gm::string cCallFlavor;
public:
        CDRInfo& operator=(const CDRInfo& other);
        inline void SetServiceName(const gm::string& ServiceName);
        inline const gm::string& GetServiceName(void);
        inline void SetBillingAccount(const gm::string& BillingAccount);
        inline const gm::string& GetBillingAccount(void);
        inline void SetCallFlavor(const gm::string& CallFlavor);
        inline const gm::string& GetCallFlavor(void);
        void Serialize(VPPMessage *);
        void Deserialize(VPPMessage *);
};
inline void
CDRInfo::SetServiceName(const gm::string& ServiceName)
{
        cServiceName = ServiceName.c_str();
}
inline const gm::string&
CDRInfo::GetServiceName(void)
{
        return cServiceName;
}
inline void
CDRInfo::SetBillingAccount(const gm::string& BillingAccount)
{
        cBillingAccount = BillingAccount.c_str();
}
inline const gm::string&
CDRInfo::GetBillingAccount(void)
{
        return cBillingAccount;
}
inline void
CDRInfo::SetCallFlavor(const gm::string& CallFlavor)
{
        cCallFlavor = CallFlavor.c_str();
```

```
}
inline const gm::string&
CDRInfo::GetCallFlavor(void)
{
        return cCallFlavor;
}
#endif
```

Document ID: 0.0.0.0.0.0.0.0

```
/*
 * CDRInfoNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:08 $
 */
#ifndef CDRINFONTFY_H
#define CDRINFONTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
#include "CDRInfo.h"
class __declspec(dllexport) CDRInfoNotification : public OneCallTokenMessageBase
{
        CDRInfo cCDRInfo;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(CDRInfoNotification,
                OneCallTokenMessageBase);
        inline CDRInfoNotification(CallToken& token, CDRInfo& cdrInfo);
        inline CDRInfo& GetCDRInfo(void);
};
inline
CDRInfoNotification::CDRInfoNotification(CallToken& token, CDRInfo& cdrInfo)
        : OneCallTokenMessageBase(M_NTFY_CDRINFO, token), cCDRInfo(cdrInfo)
{
}
inline CDRInfo&
CDRInfoNotification::GetCDRInfo(void)
{
        return cCDRInfo;
}
#endif
```

```
/*
 * CdrLogClient.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:20:05 $
 */
#ifndef CDRLOGCLIENT_H
#define CDRLOGCLIENT_H
#include "VPPMsg/CDRInfo.h"
#include "VPPMsg/Address.h"
class CEventLogClient;
enum CDRLOG_STATUS
{
        CDRLOG_SUCCESS = 0,
        CDRLOG_INFO_SERVICE_ERROR,
        CDRLOG_INFO_ACCT_ERROR,
        CDRLOG_INFO_CALLTYPE_ERROR,
        CDRLOG_FILE_ERROR
};
class __declspec(dllexport) CDRLogClient
{
public:
        CDRLogClient();
        ~CDRLogClient();
        CDRLOG_STATUS StartCall(CDRInfo *pCdrInfo, Address *Target, Address *CallingNumber);
        CDRLOG_STATUS EndCall();
        static const char AgentNotificationCall[];          // "agent_notification"
        static const char InboundCall[];                    // ""
        static const char OutboundCall[];                   // ""
private:
        CEventLogClient *m_pEventLogClient;
        CDRInfo *m_pCdrInfo;
};
#endif
```

```
/*
 * Channel.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.25 $
 * $Date: 2000/12/07 03:20:05 $
 */
#ifndef CHANNEL_H
#define CHANNEL_H
#define ENABLE_PERF_MEASURE
#include <list>
#include "VPPMsg/VPPMsg.h"
#include "VPPMsg/CallToken.h"
#include "VppUtils/Synchronize.h"
#include "VppUtils/IDLog.h"
class ServiceProvider;
class SessionNode;
class Property;
class __declspec(dllexport) Channel : public IDLog
{
public:
        typedef enum
        {
                CT_Telephony,
                CT_Media,
                CT_Speech,
                CT_TTS
        } ChannelType;
        typedef enum
        {
                S_Idle,
                S_InSession,
                S_Settling
        } ChannelState;
private:
        ChannelType cType;
        ChannelState cState;
        bool cRequiresSettlement;
        CriticalSection cStateLock;
protected:
        int     cID;
        ServiceProvider *cServiceProvider;
        CallToken cCurrentCallToken;
        SessionNode *cSession;
        void NotifySettled(void);
        bool VerifySession(SessionNode *pSession);
public:
        inline Channel(ServiceProvider *pServiceProvider, ChannelType type,
                const char *logType, int id, bool RequiresSettlement = false);
        virtual ~Channel(void);
        inline CallToken& GetCurrentCallToken(void);
```

```
        inline ChannelType GetType(void);
    inline int GetID();
        virtual void NotifyBoundToSession(SessionNode *pSessionNode);
        virtual void NotifyBoundToCall(CallToken& callToken);
        virtual void NotifyPropertySet(Property& prop);
        virtual bool NotifyCallAboutToBeDropped(SessionNode* session, CallToken& callToken);
        inline ChannelState GetState(void);
        bool SetState(Channel::ChannelState state);
        virtual bool Shutdown(void);
};
inline
Channel::Channel(ServiceProvider *pServiceProvider, Channel::ChannelType type,
        const char *logType, int id, bool RequiresSettlement)
        : cSession(0), IDLog(logType), cID(id)
{
        setLogID("<%d>", cID);
        cServiceProvider = pServiceProvider;
        cType = type;
        cState = S_Idle;
        cRequiresSettlement = RequiresSettlement;
}
inline CallToken&
Channel::GetCurrentCallToken(void)
{
        return cCurrentCallToken;
}
inline int
Channel::GetID(void)
{
        return cID;
}
inline Channel::ChannelType
Channel::GetType(void)
{
        return cType;
}
inline Channel::ChannelState
Channel::GetState(void)
{
        Synchronize sync(cStateLock);
        return cState;
}
#endif
```

```
/*
 * CmdLine.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:22:20 $
 */
#ifndef CMDLINE_H
#define CMDLINE_H
#include <list>
#include "GMString"
#pragma warning(disable:4786)
class __declspec(dllexport) CommandLine
{
        typedef struct _CommandLineNode
        {
                gm::string cAttr;
                gm::string cValue;
        } CommandLineNode;
        template class __declspec(dllexport) std::list<CommandLineNode *>;
        std::list<CommandLineNode *> cNodes;
        void KillNodes(void);
public:
        inline CommandLine(void);
        inline CommandLine(int ac, char **av);
        inline ~CommandLine(void);
        void Setup(int ac, char **av);
        const gm::string& GetValue(const gm::string& attr);
        bool DoesAttrAppear(const gm::string& attr);
};
inline
CommandLine::CommandLine(void)
{
}
inline
CommandLine::CommandLine(int ac, char **av)
{
        Setup(ac, av);
}
inline
CommandLine::~CommandLine(void)
{
        KillNodes();
}

#endif
```

```
/*
 * CMSDefs.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:20:52 $
 */
#ifndef CMSDEFS_H
#define CMSDEFS_H
#define INVALID_CMDID ((unsigned short)-1)
#define C_CMDID_START ((unsigned short)0)
#define C_CMD_START (C_CMDID_START)
#define C_RPLY_START (C_CMD_START + 100)
#define C_NTFY_START (C_RPLY_START + 100)
#define C_USER_CMDID_START (C_NTFY_START + 100)
#define C_CMD_REQUESTPINGPERIOD (C_CMD_START)
#define C_NTFY_PING (C_NTFY_START)
#define C_NTFY_PINGFAILURE (C_NTFY_START + 1)
#define C_NTFY_SETPINGPERIOD (C_NTFY_START + 2)
#endif
```

Document 2 - Page 25 of 25

```
/*
 * CMSEndpoint.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.9 $
 * $Date: 2000/12/07 03:20:52 $
 */
#ifndef CMSENDPOINT_H
#define CMSENDPOINT_H
#include "CMSMsg.h"
#include "CMSMsgQueue.h"
#include "CMSEventArray.h"
#include "CMSMsgMap.h"
#include "VPPUtils/CriticalSection.h"
class __declspec(dllexport) CMSEndpoint
{
        friend class CMSServer;
        CriticalSection cCriticalSection;
        CMSMessageQueue cMsgQueue;
        CMSMessageMap cMsgMap;
        CMSEventArray cEvents;
        unsigned long cReadThread;
        unsigned long cDispatchThread;
        unsigned long cSendPingThread;
        unsigned long cRecvPingThread;
        unsigned long cPingPeriod;
        unsigned short cPingDecayLevel;
        unsigned short cFlags;
        enum
        {
                F_AUTODISPATCH = 0x1,
                F_ISCLIENT = 0x2
        };
        static unsigned __stdcall _ReadThreadFunc(void *);
        unsigned ReadThreadFunc(void);
        static unsigned __stdcall _DispatchThreadFunc(void *);
        unsigned DispatchThreadFunc(void);
        static unsigned __stdcall _SendPingThreadFunc(void *);
        unsigned SendPingThreadFunc(void);
        static unsigned __stdcall _RecvPingThreadFunc(void *);
        unsigned RecvPingThreadFunc(void);
        void ResetSendPingTimer(void);
        void ResetReceivePingTimer(bool ResetDecay);
protected:
        IPCEndpoint *cEndpoint;
        CMSMessage* TransformMessage(CMSMessage& msg);
        virtual CMSMessage *AllocateSubclassMessage(CMSMessage *pMsg);
        virtual bool NotifyPeerConnected(void);
        virtual bool NotifyPeerDisconnected(bool FarEnd);
        virtual MESSAGEHANDLER FindMessageHandler(unsigned short cmdid,
                void **userdata);
```

```
        void SetIsClient(bool flag);
        bool IsClient(void);
        void ShutdownThreads(void);
        void EnqueueMessage(CMSMessage *pMsg);
public:
        CMSEndpoint(IPCEndpoint *pIPCEndpoint = 0);
        virtual ~CMSEndpoint(void);
        inline void SetIPCEndpoint(IPCEndpoint *pIPCEndpoint);
        inline IPCEndpoint *GetIPCEndpoint(void);

        virtual CMSMessage::SerialNumberT WriteMessage(CMSMessage& msg,
                unsigned short NumUserObjects = 0, unsigned long *UserObjects = 0);
        virtual CMSMessage *ReadMessage(unsigned short NumUserObjects = 0,
                unsigned long *UserObjects = 0);
        virtual CMSMessage *TransactMessage(CMSMessage& sendmsg);
        // Message polling facilities for non-AutoDispatch endpoints
        //
        bool MessagesWaiting(void);
        CMSMessage *GetNextMessage(void);
        inline unsigned long GetMsgInQueueEvent(void);
        // General message-handling method for AutoDispatch endpoints
        //
        virtual bool ProcessMessage(CMSMessage *pMsg);
        // Message handler management routines
        //
        inline bool AddMessageHandler(unsigned short cmdid, MESSAGEHANDLER handler,
                void *userdata = 0);
        unsigned long GetPingPeriod(void);
        void SetPingPeriod(unsigned long PeriodMilliseconds);
        void SetAutoDispatch(bool flag);
        bool IsAutoDispatch(void);
};
inline bool
CMSEndpoint::AddMessageHandler(unsigned short cmdid, MESSAGEHANDLER handler,
        void *userdata)
{
        return cMsgMap.AddMessageHandler(cmdid, handler, userdata);
}
inline unsigned long
CMSEndpoint::GetMsgInQueueEvent(void)
{
        return cMsgQueue.GetMsgInQueueEvent();
}
inline void
CMSEndpoint::SetIPCEndpoint(IPCEndpoint *pIPCEndpoint)
{
        cEndpoint = pIPCEndpoint;
}
inline IPCEndpoint *
CMSEndpoint::GetIPCEndpoint(void)
{
        return cEndpoint;
}
#endif
```

```
/*
 * CMSEventArray.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:20:52 $
 */
#ifndef CMSEVENTARRAY_H
#define CMSEVENTARRAY_H
class __declspec(dllexport) CMSEventArray
{
        unsigned short cNumEvents;
        unsigned long *cEvents;
        unsigned long cDispatchOKEvent;
        unsigned long cPingSendTimerEvent;
        unsigned long cPingRecvTimerEvent;
public:
        CMSEventArray(void);
        ~CMSEventArray(void);
        void Reset(void);
        inline unsigned long GetInTransactionEvent(void);
        inline unsigned long *GetInTransactionEventPtr(void);

        inline unsigned long GetTerminateThreadsEvent(void);
        inline unsigned long *GetTerminateThreadsEventPtr(void);
        inline unsigned long GetDispatchOKEvent(void);
        inline unsigned long GetPingSendTimerEvent(void);
        inline unsigned long GetPingReceiveTimerEvent(void);

        inline unsigned short GetNumEvents(void);
        inline unsigned long *GetEvents(void);
};
inline unsigned long
CMSEventArray::GetInTransactionEvent(void)
{
        return cEvents[0];
}
inline unsigned long *
CMSEventArray::GetInTransactionEventPtr(void)
{
        return &cEvents[0];
}
inline unsigned long
CMSEventArray::GetTerminateThreadsEvent(void)
{
        return cEvents[1];
}
inline unsigned long *
CMSEventArray::GetTerminateThreadsEventPtr(void)
{
        return &cEvents[1];
```

```
}
inline unsigned short
CMSEventArray::GetNumEvents(void)
{
        return cNumEvents;
}
inline unsigned long *
CMSEventArray::GetEvents(void)
{
        return cEvents;
}
inline unsigned long
CMSEventArray::GetDispatchOKEvent(void)
{
        return cDispatchOKEvent;
}
inline unsigned long
CMSEventArray::GetPingSendTimerEvent(void)
{
        return cPingSendTimerEvent;
}
inline unsigned long
CMSEventArray::GetPingReceiveTimerEvent(void)
{
        return cPingRecvTimerEvent;
}
#endif
```

```
/*
 * CMSMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:20:53 $
 */
#ifndef CMSMESSAGE_H
#define CMSMESSAGE_H
#include "CMSDefs.h"
#include "VPPUtils/CriticalSection.h"
#include "VPPUtils/GMString"
class IPCEndpoint;
class __declspec(dllexport) CMSMessage
{
public:
        typedef unsigned short SerialNumberT;
private:
        friend class CMSEndpoint;
        friend class CMSMessage;
        static CriticalSection cCriticalSection;
        static unsigned short cSerial;
        void AssignSerialNo(void);
        inline void init(void);
        void DeleteData(void);
        void CheckExpansion(unsigned long);
        unsigned long cAllocated;
protected:
        unsigned short cFlags;
        SerialNumberT cSerialNo;
        SerialNumberT cReplyToSerialNo;
        unsigned short cCommandID;
        unsigned long cUserdata;
        unsigned long cDataLength;
        char *cData;
        char *cParse;
        enum
        {
                F_VALID = 0x8000
        };
        inline bool Parsed(void);
        virtual void Serialize(void);
        virtual void Deserialize(void);
        void ReadFromPtr(char **);
public:
        inline CMSMessage(void);
        inline CMSMessage(unsigned short cmdid);
        inline CMSMessage(const CMSMessage& other);
        inline CMSMessage(CMSMessage *pOther);
        virtual ~CMSMessage(void);
        CMSMessage& operator=(const CMSMessage& other);
```

```
        bool ReadFrom(IPCEndpoint *pEndpoint,
                unsigned short NumUserObjects = 0, unsigned long *UserObjects = 0);

        bool WriteTo(IPCEndpoint *pEndpoint,
                unsigned short NumUserObjects = 0, unsigned long *UserObjects = 0);

        inline void SetValid(void);
        inline bool IsValid(void);
        inline void SetReplyToSerialNo(SerialNumberT);
        inline SerialNumberT GetReplyToSerialNo(void);
        inline void SetReplyTo(CMSMessage& msg);
        inline SerialNumberT GetSerialNo(void);

        inline unsigned short GetCommandID(void);
        inline void SetCommandID(unsigned short id);
        inline unsigned long GetUserdata(void);
        inline void SetUserdata(unsigned long userdata);
        inline unsigned long GetDataLength(void);
        inline char *GetData(void);
        void InsertUchar(unsigned char, char ** = 0);
        void InsertUshort(unsigned short, char ** = 0);
        void InsertUlong(unsigned long, char ** = 0);
        void InsertPtr(void *, char ** = 0);
        void InsertString(const char *, char ** = 0);
        inline void InsertString(const gm::string& str, char ** = 0);
        void InsertMessage(CMSMessage *, char ** = 0);
        void InsertRaw(const char *, unsigned long, char ** = 0);
        unsigned char ExtractUchar(char ** = 0);
        unsigned short ExtractUshort(char ** = 0);
        unsigned long ExtractUlong(char ** = 0);
        void *ExtractPtr(char ** = 0);
        void ExtractString(char *str, unsigned long maxlen, char ** = 0);
        void ExtractString(gm::string& str, char ** = 0);
        void ExtractMessage(CMSMessage *, char ** = 0);
        void ExtractRaw(char *, unsigned long len, char ** = 0);
};
inline unsigned short
CMSMessage::GetCommandID(void)
{
        return cCommandID;
}
inline void
CMSMessage::SetCommandID(unsigned short id)
{
        cCommandID = id;
}
inline void
CMSMessage::init(void)
{
        cFlags = 0;
        cCommandID = INVALID_CMDID;
        cDataLength = 0;
        cData = 0;
        cParse = 0;
        cAllocated = 0;
        cSerialNo = 0;
```

```
        cReplyToSerialNo = 0;
        cUserdata = 0;
}
inline
CMSMessage::CMSMessage(void)
{
        init();
}
inline
CMSMessage::CMSMessage(unsigned short cmdid)
{
        init();
        SetCommandID(cmdid);
}
inline
CMSMessage::CMSMessage(const CMSMessage& other)
{
        *this = other;
}
inline
CMSMessage::CMSMessage(CMSMessage *pOther)
{
        *this = *pOther;
}
inline unsigned long
CMSMessage::GetDataLength(void)
{
        return cDataLength;
}
inline char *
CMSMessage::GetData(void)
{
        return cData;
}
inline CMSMessage::SerialNumberT
CMSMessage::GetSerialNo(void)
{
        return cSerialNo;
}
inline void
CMSMessage::SetReplyToSerialNo(CMSMessage::SerialNumberT serial)
{
        cReplyToSerialNo = serial;
}
inline CMSMessage::SerialNumberT
CMSMessage::GetReplyToSerialNo(void)
{
        return cReplyToSerialNo;
}
inline unsigned long
CMSMessage::GetUserdata(void)
{
        return cUserdata;
}
inline void
CMSMessage::SetUserdata(unsigned long userdata)
```

```
{
        cUserdata = userdata;
}
inline void
CMSMessage::SetReplyTo(CMSMessage& msg)
{
        SetReplyToSerialNo(msg.GetSerialNo());
        SetUserdata(msg.GetUserdata());
}
inline void
CMSMessage::SetValid(void)
{
        cFlags |= F_VALID;
}
inline bool
CMSMessage::IsValid(void)
{
        return (bool)((cFlags & F_VALID) != 0);
}
inline bool
CMSMessage::Parsed(void)
{
        return (bool)((cParse == cData));
}
inline void
CMSMessage::InsertString(const gm::string& str, char **ptr)
{
        InsertString(str.c_str(), ptr);
}
#define DECLARE_CMSMESSAGE_SUBCLASS(Subclass, Baseclass)\
        inline Subclass(CMSMessage *pMsg) : Baseclass(pMsg) {};
#endif
```

```
/*
 * CMSMsgMap.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:20:53 $
 */
#ifndef CMSMSGMAP_H
#define CMSMSGMAP_H
#include "VPPUtils/CriticalSection.h"
#include <list>
#pragma warning(disable:4786)
typedef bool (*MESSAGEHANDLER)(class CMSMessage *, class CMSEndpoint *, void *);
class __declspec(dllexport) CMSMessageMap
{
        struct MessageMapEntry
        {
                unsigned short cCommandID;
                MESSAGEHANDLER cHandler;
                void *cUserdata;
        };

        CriticalSection cCriticalSection;
        template class __declspec(dllexport) std::list<MessageMapEntry *>;
        std::list<MessageMapEntry *> cMap;
        MessageMapEntry *FindEntry(unsigned short cmdid);
public:
        inline CMSMessageMap(void);
        ~CMSMessageMap(void);
        bool AddMessageHandler(unsigned short cmdid, MESSAGEHANDLER handler,
                void *userdata);

        MESSAGEHANDLER FindMessageHandler(unsigned short cmdid, void **userdata);
};
inline
CMSMessageMap::CMSMessageMap(void)
{
}

#endif
```

```
/*
 * CMSMsgQueue.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:20:53 $
 */
#ifndef CMSMSGQUEUE_H
#define CMSMSGQUEUE_H
#include "VPPUtils/CriticalSection.h"
#include <list>
#pragma warning(disable:4786)
class CMSMessage;
class __declspec(dllexport) CMSMessageQueue
{
        template class __declspec(dllexport) std::list<CMSMessage *>;
        CriticalSection cCriticalSection;
        std::list<CMSMessage *> cMsgs;
        unsigned long cMsgInQueueEvent;
        void RemoveMsg(CMSMessage *pMsg, bool DeleteMsg);
public:
        CMSMessageQueue(void);
        ~CMSMessageQueue(void);
        bool InsertMessage(CMSMessage *pMsg);

        bool IsMessageWaiting(CMSMessage **ppMsg = 0, bool Remove = false);
        CMSMessage *RemoveMessage(unsigned short NumUserEvents = 0, unsigned long *UserEvents = 0);
        inline unsigned long GetMsgInQueueEvent(void);
        void Clear(void);
};
inline unsigned long
CMSMessageQueue::GetMsgInQueueEvent(void)
{
        return cMsgInQueueEvent;
}
#endif
```

```
/*
 * CMSMultiClient.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:20:53 $
 */
#ifndef CMSMULTICLIENT_H
#define CMSMULTICLIENT_H
#include "VPPUtils/CriticalSection.h"
#include "CMSClient.h"
#include <list>
#pragma warning(disable:4786)
#pragma warning(disable:4251)
class __declspec(dllexport) CMSMultiClient
{
protected:
        template class __declspec(dllexport) std::list<CMSClient *>;

        CriticalSection cCriticalSection;
        std::list<CMSClient *> cClients;
private:
        std::list<CMSClient *>::iterator cIter;
        void ResetIterator(void);
        void IncrementIterator(void);
public:
        CMSMultiClient(void);
        virtual ~CMSMultiClient(void);
        void AddClient(CMSClient *pClient);
        void RemoveClient(CMSClient *pClient);
        inline CriticalSection& GetNodeListLock(void);
        unsigned short GetNumClients(void);
        CMSClient *GetClient(unsigned short index);
        CMSClient *GetNextClient(void);
        CMSMessage::SerialNumberT BroadcastMessage(CMSMessage& msg);
};
inline CriticalSection&
CMSMultiClient::GetNodeListLock(void)
{
        return cCriticalSection;
}
#endif
```

```
/*
 * CMSServer.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.6 $
 * $Date: 2000/12/07 03:20:53 $
 */
#ifndef CMSSERVER_H
#define CMSSERVER_H
#include "CMSEndpoint.h"
#include "VPPUtils/CriticalSection.h"
#include "CMSServerNode.h"
#include <list>
#pragma warning(disable:4786)
class __declspec(dllexport) CMSServer : public CMSEndpoint
{
        template class __declspec(dllexport) std::list<CMSServerNode *>;
        friend class CMSServerNode;

        CriticalSection cCriticalSection;
        std::list<CMSServerNode *> cNodes;
        bool NewConnection(IPCEndpoint *pEndpoint);
        bool NotifyServerNodeDisconnected(CMSServerNode *pServerNode);

public:
        CMSServer(IPCEndpoint *pIPCEndpoint = 0);
        virtual ~CMSServer(void);
        bool Listen(void);
        bool ClientConnected(void);
        int WaitForClientConnect(unsigned short NumUserObjects = 0,
                unsigned long *UserObjects = 0);
        // Client statistics
        //
        inline CriticalSection& GetNodeListLock(void);
        unsigned short GetNumOpenConnections(void);
        CMSServerNode *GetServerNode(unsigned short index);
        virtual CMSServerNode *CreateServerNode(IPCEndpoint *pIPCEndpoint);
        CMSMessage::SerialNumberT BroadcastMessage(CMSMessage& msg);
};
inline CriticalSection&
CMSServer::GetNodeListLock(void)
{
        return cCriticalSection;
}
#endif
```

```
/*
 * CMSServerNode.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:20:54 $
 */
#ifndef CMSSERVERNODE_H
#define CMSSERVERNODE_H
#include "IPCEndpoint.h"
#include "CMSEndpoint.h"
class CMSServer;
class __declspec(dllexport) CMSServerNode : public CMSEndpoint
{
        virtual MESSAGEHANDLER FindMessageHandler(unsigned short cmdid,
                void **userdata);

protected:
        CMSServer *cServer;
        virtual bool NotifyPeerDisconnected(bool FarEnd);
public:
        inline CMSServerNode(CMSServer *pServer, IPCEndpoint *pIPCEndpoint = 0);
        virtual ~CMSServerNode(void);
        inline CMSServer *GetServer(void);
};
inline
CMSServerNode::CMSServerNode(CMSServer *pServer, IPCEndpoint *pIPCEndpoint)
        : CMSEndpoint(pIPCEndpoint)
{
        cServer = pServer;
        SetAutoDispatch(true);
}
inline CMSServer *
CMSServerNode::GetServer(void)
{
        return cServer;
}
#endif
```

```
/*
 * CNGDetectedNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:08 $
 */
#ifndef CNGDETECTEDNTFY_H
#define CNGDETECTEDNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) CNGDetectedNotification : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(CNGDetectedNotification,
                OneCallTokenMessageBase);
        inline CNGDetectedNotification(CallToken& token);
};
inline __declspec(dllexport)
CNGDetectedNotification::CNGDetectedNotification(CallToken& token)
        : OneCallTokenMessageBase(M_NTFY_CNGDETECTED, token)
{
}
#endif
```

```
/*
 * CompositeMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:11 $
 */
#ifndef COMPOSITEMSG_H
#define COMPOSITEMSG_H
#include <list>
#include "VPPDefs.h"
#include "VPPMsg.h"
#pragma warning(disable:4786)
class __declspec(dllexport) CompositeMessage : public VPPMessage
{
        template class __declspec(dllexport) std::list<VPPMessage *>;
        std::list<VPPMessage *> cMessages;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(CompositeMessage, VPPMessage);
        inline CompositeMessage(void);
        ~CompositeMessage(void);
        inline void AddMessage(VPPMessage *pMsg);
        inline std::list<VPPMessage *>& GetMessages(void);
};
inline
CompositeMessage::CompositeMessage(void)
        : VPPMessage(M_CMD_COMPOSITEMSG)
{
}
inline void
CompositeMessage::AddMessage(VPPMessage *pMsg)
{
        cMessages.push_back(pMsg);
}
inline std::list<VPPMessage *>&
CompositeMessage::GetMessages(void)
{
        return cMessages;
}
#endif
```

```
/*
 * CompositePromptCreatedNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:12 $
 */
#ifndef COMPOSITEPROMPTCREATEDNTFY_H
#define COMPOSITEPROMPTCREATEDNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) CompositePromptCreatedNotification
        : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(CompositePromptCreatedNotification,
                OneCallTokenMessageBase);
        inline CompositePromptCreatedNotification(CallToken& token);
};
inline
CompositePromptCreatedNotification::CompositePromptCreatedNotification(
        CallToken& token)
        : OneCallTokenMessageBase(M_NTFY_COMPOSITEPROMPTCREATED, token)
{
}
#endif
```

```
/*
 * CompositePromptEndedNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:12 $
 */
#ifndef COMPOSITEPROMPTENDEDNTFY_H
#define COMPOSITEPROMPTENDEDNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) CompositePromptEndedNotification
        : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(CompositePromptEndedNotification,
                OneCallTokenMessageBase);
        inline CompositePromptEndedNotification(CallToken& token);
};
inline
CompositePromptEndedNotification::CompositePromptEndedNotification(
        CallToken& token)
        : OneCallTokenMessageBase(M_NTFY_COMPOSITEPROMPTENDED, token)
{
}
#endif
```

```
/*
 * CompositePromptPlayDoneNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:12 $
 */
#ifndef COMPOSITEPROMPTPLAYDONENTFY_H
#define COMPOSITEPROMPTPLAYDONENTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) CompositePromptPlayDoneNotification
        : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(CompositePromptPlayDoneNotification,
                OneCallTokenMessageBase);
        inline CompositePromptPlayDoneNotification(CallToken& token);
};
inline
CompositePromptPlayDoneNotification::CompositePromptPlayDoneNotification(
        CallToken& token)
        : OneCallTokenMessageBase(M_NTFY_COMPOSITEPROMPTPLAYDONE, token)
{
}
#endif
```

```
/*
 * CompositePromptPlayingNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:12 $
 */
#ifndef COMPOSITEPROMPTPLAYINGNTFY_H
#define COMPOSITEPROMPTPLAYINGNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) CompositePromptPlayingNotification
        : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(CompositePromptPlayingNotification,
                OneCallTokenMessageBase);
        inline CompositePromptPlayingNotification(CallToken& token);
};
inline
CompositePromptPlayingNotification::CompositePromptPlayingNotification(
        CallToken& token)
        : OneCallTokenMessageBase(M_NTFY_COMPOSITEPROMPTPLAYING, token)
{
}
#endif
```

```
/*
 * CompositePromptSegmentAppendedNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:12 $
 */
#ifndef COMPOSITEPROMPTSEGMENTAPPENDEDNTFY_H
#define COMPOSITEPROMPTSEGMENTAPPENDEDNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) CompositePromptSegmentAppendedNotification
        : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(CompositePromptSegmentAppendedNotification,
                OneCallTokenMessageBase);
        inline CompositePromptSegmentAppendedNotification(CallToken& token);
};
inline
CompositePromptSegmentAppendedNotification::CompositePromptSegmentAppendedNotification(
        CallToken& token)
        : OneCallTokenMessageBase(M_NTFY_COMPOSITEPROMPTSEGMENTAPPENDED, token)
{
}
#endif
```

Document ID: 0.0.2.2.1.0.0.0.0

```
/*
 * CompositePromptSegmentPlayDoneNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:13 $
 */
#ifndef COMPOSITEPROMPTSEGMENTPLAYDONENTFY_H
#define COMPOSITEPROMPTSEGMENTPLAYDONENTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) CompositePromptSegmentPlayDoneNotification
        : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(CompositePromptSegmentPlayDoneNotification,
                OneCallTokenMessageBase);
        inline CompositePromptSegmentPlayDoneNotification(CallToken& token);
};
inline
CompositePromptSegmentPlayDoneNotification::CompositePromptSegmentPlayDoneNotification(
        CallToken& token)
        : OneCallTokenMessageBase(M_NTFY_COMPOSITEPROMPTSEGMENTPLAYDONE, token)
{
}
#endif
```

```
/*
 * CompositePromptSegmentPlayingNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:13 $
 */
#ifndef COMPOSITEPROMPTSEGMENTPLAYINGNTFY_H
#define COMPOSITEPROMPTSEGMENTPLAYINGNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) CompositePromptSegmentPlayingNotification
        : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(CompositePromptSegmentPlayingNotification,
                OneCallTokenMessageBase);
        inline CompositePromptSegmentPlayingNotification(CallToken& token);
};
inline
CompositePromptSegmentPlayingNotification::CompositePromptSegmentPlayingNotification(
        CallToken& token)
        : OneCallTokenMessageBase(M_NTFY_COMPOSITEPROMPTSEGMENTPLAYING, token)
{
}
#endif
```

Document ID: 2000-1206-0

```
/* CONFIG.H */
#define HAVE_STRING_H
#define STDC_HEADERS
#define REGEX_MALLOC
```

Document provided by IHS

```
/*
 * ConfigCmdLineList.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:22:20 $
 */
#ifndef CONFIGCMDLINELIST_H
#define CONFIGCMDLINELIST_H
#include "ConfigList.h"
#pragma warning(disable:4786)
class __declspec(dllexport) ConfigCmdLineList : public ConfigList
{
public:
        void Initialize() { };
        void Initialize(int ac, char **av);
};
#endif
```

Document Page 15 of 15

```
/*
 * ConfigFile.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:22:21 $
 */
#ifndef CONFIGFILE_H
#define CONFIGFILE_H
#include <list>
#include "GMString"
#include "Timer.h"
#include "ConfigFileList.h"
#pragma warning(disable:4786)
//class __declspec(dllexport) ConfigFile : public ConfigFileList
class __declspec(dllexport) ConfigFile : public ConfigList
{
        ConfigFileList fileList[2];
        int     currentListIndex;
        bool initialized;
        Timer::TimerID timerID;
        gm::string      configFile;
        long int lastModTime;
        unsigned long configRefreshRate;
        void ConfigFile::UpdateConfig();
public:
        ConfigFile();
        ConfigFile(const char *filename);
        ~ConfigFile();
        void Initialize();
        unsigned short GetNumLines(void);
        const char *GetLine(unsigned short ndx);    // Use for non-refreshed only
        bool GetString(const char *key, gm::string& retString);
};
#endif
```

```
/*
 * ConfigFileList.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:22:21 $
 */
#ifndef CONFIGFILELIST_H
#define CONFIGFILELIST_H
#include "GMString"
#include "ConfigList.h"
#pragma warning(disable:4786)
class __declspec(dllexport) ConfigFileList : public ConfigList
{
public:
        void Initialize();
        void Initialize(const char *filename);
};
#endif
```

Exhibit 2002 - Page 1 of 2

```
/*
 * ConfigInternalList.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:22:21 $
 */
#ifndef CONFIGINTERNALLIST_H
#define CONFIGINTERNALLIST_H
#include "ConfigList.h"
#pragma warning(disable:4786)
class __declspec(dllexport) ConfigInternalList : public ConfigList
{
        static char *configString[];
public:
        void Initialize();
};
#endif
```

```
/*
 * ConfigList.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:22:21 $
 */
#ifndef CONFIGLIST_H
#define CONFIGLIST_H
#include <list>
#include "GMString"
#include "Synchronized.h"
#pragma warning(disable:4786)
class __declspec(dllexport) ConfigList
{
private:
        template class __declspec(dllexport) std::list<char *>;
protected:
        std::list<char *> list;
public:
        ConfigList(void);
        ~ConfigList(void);
        virtual void Initialize(void) = 0;
        void CleanList();
        unsigned short GetNumLines(void);
        const char *GetLine(unsigned short ndx);
        bool GetString(const char *key, gm::string& retString);
};
#endif
```

```
/*
 * ConfigMgr.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:22:22 $
 */
#ifndef CONFIGMGR_H
#define CONFIGMGR_H
#include "ConfigInternalList.h"
#include "ConfigCmdLineList.h"
#include "ConfigFile.h"
#pragma warning(disable:4786)
#include "GMString"
//#define USE_REGISTRY
#ifdef USE_REGISTRY
#define REGISTRY_BASE          "Software.General Magic."
#else
#define REGISTRY_BASE          ""
#endif
#define MTS_REGISTRY_BASE  REGISTRY_BASE"MTS"
class ConfigFile;
class __declspec(dllexport) ConfigMgr
{
public:
        static bool GetString(const char *parent, const char *key, gm::string& ret);
        static bool GetInteger(const char *parent, const char *key, int &value);
        static bool GetBoolean(const char *parent, const char *key, bool &value);
        static void Initialize(int ac, char **av);
private:
        static bool         initialized;
        static ConfigInternalList internalList;
        static ConfigCmdLineList cmdLineList;
        static ConfigFile fileList;
};
#endif
```

```
/*
 * ConnectCallsMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:13 $
 */
#ifndef CONNECTCALLSMSG_H
#define CONNECTCALLSMSG_H
#include "TwoCallTokensMsgBase.h"
class __declspec(dllexport) ConnectCallsMessage : public TwoCallTokensMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(ConnectCallsMessage, TwoCallTokensMessageBase);
        inline ConnectCallsMessage(CallToken& call1, CallToken& call2);
};
inline
ConnectCallsMessage::ConnectCallsMessage(CallToken& call1, CallToken& call2)
        : TwoCallTokensMessageBase(M_CMD_CONNECTCALLS, call1, call2)
{
}
#endif
```

```
/*
 * ConnectionListener.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:20:05 $
 */
#ifndef _CONNECTION_LISTENER_H
#define _CONNECTION_LISTENER_H
class __declspec(dllexport) ConnectionListener {
public:
   virtual void Disconnected() = 0;
   virtual ~ConnectionListener();
};
#endif
```

Document ID: 2000-1206-1

```
/*
 * ConnectionMgr.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:20:06 $
 */
#ifndef _CONNECTION_MGR_H
#define _CONNECTION_MGR_H
#include <VPPUtils/Hashtable.h>
#include <VPPUtils/Queue.h>
#include "invui_c.h"
#include "ConnectionListener.h"
class ConnectionWatcher;
class ConnectionMgr {
public:
   template ::Queue<ConnectionListener *>;
   typedef ::Queue<ConnectionListener*> ConnectionListenerList;
   template ::Hashtable<char*, ConnectionListenerList*>;
   typedef ::Hashtable<char*, ConnectionListenerList*>
      ConnectionTable;
   static void addConnectionListener(invui::Pingable_ptr ping,
                                     ConnectionListener* listener);
   static void removeConnectionListener(ConnectionListener* listener);
   static void shutdown(char* id);
private:
   friend ConnectionWatcher;
   static ConnectionTable* table;
};
#endif
```

```
/*
 * ConnectionWatcher.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:20:06 $
 */
#ifndef _CONNECTION_WATCHER_H
#define _CONNECTION_WATCHER_H
#include <VPPUtils/Thread.h>
#include "invui_c.h"
class ConnectionWatcher : public Thread {
   invui::Pingable_var  ping;
   char* id;
public:
   ConnectionWatcher(invui::Pingable_ptr p, char* id);
   ~ConnectionWatcher();
   virtual void run();
};
#endif
```

```
/*
 * CreateCompositePromptMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:13 $
 */
#ifndef CREATECOMPOSITEPROMPTMSG_H
#define CREATECOMPOSITEPROMPTMSG_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) CreateCompositePromptMessage
        : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(CreateCompositePromptMessage,
                OneCallTokenMessageBase);
        inline CreateCompositePromptMessage(CallToken& token);
};
inline
CreateCompositePromptMessage::CreateCompositePromptMessage(
        CallToken& token)
        : OneCallTokenMessageBase(M_CMD_CREATECOMPOSITEPROMPT, token)
{
}
#endif
```

```
/*
 * CreateDynamicGrammarMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:21:14 $
 */
#ifndef CREATEDYNAMICGRAMMARMSG_H
#define CREATEDYNAMICGRAMMARMSG_H
#include "DynamicGrammarMsg.h"
#include "VPPUtils/GMString"
class __declspec(dllexport) CreateDynamicGrammarMessage : public DynamicGrammarMessage
{
        gm::string cGSL;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(CreateDynamicGrammarMessage, DynamicGrammarMessage);
        inline CreateDynamicGrammarMessage(CallToken& token, const gm::string& db_key,
                const gm::string& gsl);
        inline gm::string& GetGSL(void);
};
inline
CreateDynamicGrammarMessage::CreateDynamicGrammarMessage(CallToken& token,
        const gm::string& db_key, const gm::string& gsl)
        : DynamicGrammarMessage(M_CMD_CREATEDYNAMICGRAMMAR, token, db_key), cGSL(gsl.c_str())
{
}
inline gm::string&
CreateDynamicGrammarMessage::GetGSL(void)
{
        return cGSL;
}
#endif
```

```
/*
 * CreateDynamicGrammarResultNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:14 $
 */
#ifndef CREATEDYNAMICGRAMMARRESULTNTFY_H
#define CREATEDYNAMICGRAMMARRESULTNTFY_H
#include "DynamicGrammarResultNtfy.h"
class __declspec(dllexport) CreateDynamicGrammarResultNotification : public DynamicGrammarResultNotification
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(CreateDynamicGrammarResultNotification,
DynamicGrammarResultNotification);
        inline CreateDynamicGrammarResultNotification(CallToken& ct, const gm::string& db_key, Result result);
};
inline
CreateDynamicGrammarResultNotification::CreateDynamicGrammarResultNotification(CallToken& callToken,
        const gm::string& db_key, DynamicGrammarResultNotification::Result result)
        : DynamicGrammarResultNotification(M_NTFY_CREATEDYNAMICGRAMMARRESULT, callToken,
db_key, result)
{
}
#endif
```

```
/*
 * CriticalSection.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:22:22 $
 */
#ifndef CRITICALSECTION_H
#define CRITICALSECTION_H
#include <windows.h>
class __declspec(dllexport) CriticalSection
{
        CRITICAL_SECTION cCriticalSection;
public:
        CriticalSection(void);
        ~CriticalSection(void);
        void Enter(void);
        void Leave(void);
};
#endif
```

```
/*
 * DefaultLog.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.13 $
 * $Date: 2000/12/07 03:22:22 $
 */
#ifndef _DEFAULT_LOG_H
#define _DEFAULT_LOG_H
#include <stdio.h>
#include "Log.h"
#include "Timer.h"
#include <List>
#include "GMString"
class LogFile;
/**
 * Default Log class
 */
class __declspec(dllexport) DefaultLog : public VPPLog {
public:
   DefaultLog(const char* n);
         static void DefaultLog::Initialize(const char *filename, bool overwrite = false);
         static void DefaultLog::Shutdown();

         virtual void va_error(const char *fmt, const va_list vl);
   virtual void va_warning(const char *fmt, const va_list vl);
   virtual void va_info(const char *fmt, const va_list vl);
   virtual void va_diag(const char *fmt, const va_list vl);
   virtual void va_debug(const char *fmt, const va_list vl);
   virtual void error(const char *fmt, ...);
   virtual void warning(const char *fmt, ...);
   virtual void info(const char *fmt, ...);
   virtual void diag(const char *fmt, ...);
   virtual void debug(const char *fmt, ...);
   static bool SetOutputFilename(const char* n, bool overwrite=FALSE);
friend class DefaultLogFactory;
   void setMask(unsigned int enableMask);
   bool errorEnabled();
   bool warningEnabled();
   bool infoEnabled();
   bool diagEnabled();
   bool debugEnabled();
         void openNewFile();
         const char* name;                    // Name of log group
   void print(const char* type, const char *fmt, const va_list vl);
         static LogFile *logFile;
};
/**
 * Default Log Factory
 */
class DefaultLogFactory : public LogFactory, public Monitor {
```

```
public:
//   static void Initialize();
//   static void Shutdown();
   DefaultLogFactory();
   virtual ~DefaultLogFactory();
   virtual VPPLog* createLog(const char* name);
   void updateLogLevels();
private:
//   static Monitor monitor;
   unsigned int GetMaskFor(const char* name);
   unsigned int getRegistryMask(const char* name, unsigned int defaultMask);
   unsigned int defaultEnableMask;
   bool cAllPresent;
   gm::string currentConfig;
   std::list<DefaultLog *> logs;
   Timer::TimerID timerID;
   Hashtable<char*, unsigned int>* mask_table;
};
#endif
```

Document ID: 1E0D0D0D0D0D0D0D

```
/*
 * DeleteDynamicGrammarMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:14 $
 */
#ifndef DELETEDYNAMICGRAMMARMSG_H
#define DELETEDYNAMICGRAMMARMSG_H
#include "DynamicGrammarMsg.h"
class __declspec(dllexport) DeleteDynamicGrammarMessage : public DynamicGrammarMessage
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(DeleteDynamicGrammarMessage, DynamicGrammarMessage);
        inline DeleteDynamicGrammarMessage(CallToken& token, const gm::string& db_key);
};
inline
DeleteDynamicGrammarMessage::DeleteDynamicGrammarMessage(CallToken& token,
        const gm::string& db_key)
        : DynamicGrammarMessage(M_CMD_DELETEDYNAMICGRAMMAR, token, db_key)
{
}
#endif
```

```
/*
 * DeleteDynamicGrammarResultNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:15 $
 */
#ifndef DELETEDYNAMICGRAMMARRESULTNTFY_H
#define DELETEDYNAMICGRAMMARRESULTNTFY_H
#include "DynamicGrammarResultNtfy.h"
class __declspec(dllexport) DeleteDynamicGrammarResultNotification : public DynamicGrammarResultNotification
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(DeleteDynamicGrammarResultNotification,
DynamicGrammarResultNotification);
        inline DeleteDynamicGrammarResultNotification(CallToken& ct, const gm::string& db_key, Result result);
};
inline
DeleteDynamicGrammarResultNotification::DeleteDynamicGrammarResultNotification(CallToken& callToken,
        const gm::string& db_key, DynamicGrammarResultNotification::Result result)
        : DynamicGrammarResultNotification(M_NTFY_DELETEDYNAMICGRAMMARRESULT, callToken,
db_key, result)
{
}
#endif
```

```
/*
 * DialogicMChannel.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.28 $
 * $Date: 2000/12/07 03:25:08 $
 */
#ifndef DIALOGICMCHANNEL_H
#define DIALOGICMCHANNEL_H
#include <srllib.h>
#include <dxxxlib.h>
#include "MTSLib/MediaChannel.h"
#include "DialogicMSP.h"
class __declspec(dllexport) DialogicMChannel : public MediaChannel
{
private:
        typedef enum
        {
                S_Idle,
                S_PlayingFile,
                S_PlayingStream,
                S_PlayingTTS,
                S_PlayingSilence,
    S_PlayingDTMF,
                S_Recording,
                S_StreamRecording
        } DMChannelState;
        enum AudioFileFormat {
                eAFF_Error,
                eAFF_CantDetermine,
                eAFF_Sphere
        };
        int cHdl;
        DX_IOTT cIOTT;
        DV_TPT cTPT;
        DX_XPB cXPB;
        DX_UIO cUIO;
        DMChannelState cDMChanState;
        CriticalSection    cStateLock;
        // the following are used for stream play
        long    cCurFilePos;
        FILE    *cpPlayFile;
        HANDLE          cStreamPlayReadMutex;
        // silence prompt support
        //
        unsigned long cSilenceBytes;
        HANDLE                  cSilencePromptTimer;
        HANDLE                  cSilencePromptTimerThreadH;
        unsigned        cSilencePromptTimerThreadId;
        HANDLE                  cStopTimerEvent;                        // Signaled when NotifyEndOfData is
called
```

```
        inline MediaServiceProvider *GetMSP(void);
        bool HandleEvent(unsigned long evhandle);
        FILE *OpenFile(const char *filename, const char *mode);
        int  StreamRead(char *buf, unsigned int count);
        AudioFileFormat CheckAudioFileFormat(const gm::string& filename);
        unsigned int NumSilenceBytesToPlay(unsigned int count);
        void CleanupPlay(void);
        void SafeDeleteTTSFile(void);
        void CloseDlgcFileHandle(void);
protected:
        unsigned long cRecordStartedTime;
   unsigned long cRecordingStoppedTime;
   unsigned long cPlayingStoppedTime;
   int cBosTimeout;
   int cTmsTimeout;
   int cEosTimeout;
        //
        // From Channel
        //
        virtual unsigned long ListenTo(unsigned short timeslot);
        //
        // From MediaChannel
        //
        virtual unsigned long Play(const gm::string& filename, bool ttsSource, bool useCallback);
        virtual unsigned long PlaySilence(unsigned long durationmsecs);
   virtual unsigned long PlayDTMF(const gm::string& dtmfString);
        static int DMChanRecordWrite(int fHandle, char *buffer, unsigned int count);
        static int DMChanRecordRead(int fHandle, char *buffer, unsigned int count);
        static long DMChanRecordSeek(int fHandle, long offset, int initPos);
        static int DMChanPlayWrite(int fHandle, char *buffer, unsigned int count);
        static int DMChanPlayRead(int fHandle, char *buffer, unsigned int count);
        static long DMChanPlaySeek(int fHandle, long offset, int initPos);
        static int DMChanPlaySilenceWrite(int fHandle, char *buffer, unsigned int count);
        static int DMChanPlaySilenceRead(int fHandle, char *buffer, unsigned int count);
        static long DMChanPlaySilenceSeek(int fHandle, long offset, int initPos);
public:
#ifdef ENABLE_PERF_MEASURE
        static bool cFirstCallback;
        static unsigned long cCallbackTime;
#endif
        DialogicMChannel(ServiceProvider *pServiceProvider, const char *logType, int chanNum);
        virtual ~DialogicMChannel(void);
        bool Setup(const gm::string& chname);
        inline int GetHdl(void);
        inline DMChannelState GetState(void);
        inline void SetState(DMChannelState state);
        static unsigned __stdcall timerThreadFunc(LPVOID);
        unsigned TimerThreadFunc(void);
        static long dx_handler(unsigned long evhandle);
        virtual unsigned long Record(const gm::string& filename, bool bStreaming, int bos_timeout, int tms_timeout,
int eos_timeout); // all timeouts in 10msecs
        virtual unsigned long Stop(void);
};
inline MediaServiceProvider *
DialogicMChannel::GetMSP(void)
{
```

```
        return (MediaServiceProvider *)cServiceProvider;
}
inline int
DialogicMChannel::GetHdl(void)
{
        return cHdl;
}
inline DialogicMChannel::DMChannelState
DialogicMChannel::GetState(void)
{
        return cDMChanState;
}
inline void
DialogicMChannel::SetState(DialogicMChannel::DMChannelState state)
{
        cDMChanState = state;
}
#endif
```

```
/*
 * DialogicMSP.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.7 $
 * $Date: 2000/12/07 03:25:09 $
 */
#ifndef DIALOGICMSP_H
#define DIALOGICMSP_H
#include "MTSLib/MediaServiceProvider.h"
#include "VPPUtils/Queue.h"
#include "NotifySampleEvent.h"
class DialogicMChannel;
class NotifySampleEvent;
class __declspec(dllexport) DialogicMSP : public MediaServiceProvider
{
        static const char const *cLogType;
   static DialogicMSP *cInstance;
   int cDefaultAdjustPlayVolumeDB;
   template class __declspec(dllexport) ::Queue<NotifySampleEvent*>;
   template class __declspec(dllexport) ::BlockingQueue<NotifySampleEvent*>;
   BlockingQueue<NotifySampleEvent*> samplingQueue;
   enum QueueState {
      QUEUE_RUNNING,
      QUEUE_STOPPING,
      QUEUE_STOPPED
   } queueState;
   //  QueueState queueState;
   int numThreads;
        bool NewChannel(const char *name, int chanNum);
public:
   DialogicMSP(MTSServer *pServer);
   virtual ~DialogicMSP(void);
   virtual bool Setup(void);
   virtual bool Teardown(void);
   DialogicMChannel *FindChannelForDescriptor(int desc);
   void samplingMainLoop();
   void sendSamplingRequest(SampleListener* listener, unsigned char* bytes, unsigned int size);
   static DialogicMSP *GetInstance(void);
   static void SetInstance(DialogicMSP *ptr);
   inline int GetDefaultAdjustPlayVolumeDB(void);
};
inline int
DialogicMSP::GetDefaultAdjustPlayVolumeDB(void)
{
   return cDefaultAdjustPlayVolumeDB;
}
#endif
```

```
/*
 * DisconnectCallsMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:15 $
 */
#ifndef DISCONNECTCALLSMSG_H
#define DISCONNECTCALLSMSG_H
#include "TwoCallTokensMsgBase.h"
class __declspec(dllexport) DisconnectCallsMessage : public TwoCallTokensMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(DisconnectCallsMessage, TwoCallTokensMessageBase);
        inline DisconnectCallsMessage(CallToken& call1, CallToken& call2);
};
inline
DisconnectCallsMessage::DisconnectCallsMessage(CallToken& call1, CallToken& call2)
        : TwoCallTokensMessageBase(M_CMD_DISCONNECTCALLS, call1, call2)
{
}
#endif
```

```
/*
 * DropCallMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:21:15 $
 */
#ifndef DROPCALLMSG_H
#define DROPCALLMSG_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
#include "CDRInfo.h"
class __declspec(dllexport) DropCallMessage : public OneCallTokenMessageBase
{
        CDRInfo cCDRInfo;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(DropCallMessage, OneCallTokenMessageBase);
        inline DropCallMessage(CallToken& token, CDRInfo& cdrinfo);
        inline CDRInfo& GetCDRInfo(void);
};
inline
DropCallMessage::DropCallMessage(CallToken& token, CDRInfo& cdrinfo)
        : OneCallTokenMessageBase(M_CMD_DROPCALL, token), cCDRInfo(cdrinfo)
{
}
inline CDRInfo&
DropCallMessage::GetCDRInfo(void)
{
        return cCDRInfo;
}
#endif
```

```
/*
 * DTMFNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:14 $
 */
#ifndef DTMFNTFY_H
#define DTMFNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) DTMFNotification : public OneCallTokenMessageBase
{
        char cDTMF;
        unsigned long cTimeStamp;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(DTMFNotification,
                OneCallTokenMessageBase);
        inline DTMFNotification(CallToken& token, char dtmf, unsigned long timestamp);
        inline char GetDTMF(void);
        inline unsigned long GetTimeStamp(void);
};
inline
DTMFNotification::DTMFNotification(CallToken& token,
        char dtmf, unsigned long timestamp)
        : OneCallTokenMessageBase(M_NTFY_DTMF, token),
          cDTMF(dtmf), cTimeStamp(timestamp)
{
}
inline char
DTMFNotification::GetDTMF(void)
{
        return cDTMF;
}
inline unsigned long
DTMFNotification::GetTimeStamp(void)
{
        return cTimeStamp;
}
#endif
```

```
/*
 * DTMFSegment.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:20:06 $
 */
#ifndef DTMFSEGMENT_H
#define DTMFSEGMENT_H
#include "PromptSegment.h"
#include <VPPUtils/GMString>
class DTMFSegment : public PromptSegment
{
   gm::string cDTMF;
public:
   inline DTMFSegment(const gm::string& dtmf);
          virtual ~DTMFSegment(void);
          virtual bool Play(MediaChannel *pMediaChannel,
                CallToken& ct);
};

inline
DTMFSegment::DTMFSegment(const gm::string& dtmf)
          : PromptSegment(PromptSegment::ST_DTMF)
{
          cDTMF = dtmf.c_str();
}
#endif
```

```
/*
 * DynamicGrammar.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:20:06 $
 */
#ifndef _DYNAMIC_GRAMMAR_H
#define _DYNAMIC_GRAMMAR_H
#include <VPPUtils/GMString>
#include <MTSLib/GrammarHandle.h>
class __declspec(dllexport) DynamicGrammar {
public:
   enum GrammarType {
      TEXT,
      SET
   } cGrammarType;
   virtual GrammarType GetGrammarType() = 0;
};
class __declspec(dllexport) GrammarItem {
public:
   GrammarItem(GrammarHandle::HandleType type, gm::string& k, float w) :
      handle(type, k), weight(w) {
   }
   GrammarHandle handle;
   float weight;
};
template class __declspec(dllexport) std::list<GrammarItem*>;
class __declspec(dllexport) GrammarSet :
   public DynamicGrammar, public std::list<GrammarItem*>
{
public:
   GrammarSet() : std::list<GrammarItem*>() {}
   /**
    * Copy Constructor
    */
   GrammarSet(const GrammarSet& set) {
      std::list<GrammarItem *>::const_iterator itr = set.begin();
      while (itr != set.end()) {
         GrammarItem* item = *itr;
         add( item->handle, item->weight);
         itr ++;
      }
   }
   ~GrammarSet() {
      std::list<GrammarItem *>::iterator itr = begin();
      while (itr != end())
      {
         delete (*itr);
         itr++;
      }
```

```
  }
  void add(GrammarHandle& handle, float weight) {
    push_back(new GrammarItem(handle.cHandleType, handle.key, weight));
  }
  virtual GrammarType GetGrammarType() {
    return DynamicGrammar::SET;
  }
};
class __declspec(dllexport) TextGrammar : public DynamicGrammar {
public:
  TextGrammar(gm::string& g, gm::string& k) :
    grammar(g.c_str()), key(k.c_str()) {
  }
  virtual GrammarType GetGrammarType() {
    return DynamicGrammar::TEXT;
  }
  gm::string grammar;
  gm::string key;
};
#endif
```

```
/*
 * DynamicGrammarMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:15 $
 */
#ifndef DYNAMICGRAMMARMSG_H
#define DYNAMICGRAMMARMSG_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) DynamicGrammarMessage : public OneCallTokenMessageBase
{
        gm::string cDBKey;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(DynamicGrammarMessage, OneCallTokenMessageBase);
        inline DynamicGrammarMessage(unsigned short id, CallToken& token, const gm::string& db_key);
        inline gm::string& GetDBKey(void);
};
inline
DynamicGrammarMessage::DynamicGrammarMessage(unsigned short id, CallToken& token,
        const gm::string& db_key) : OneCallTokenMessageBase(id, token), cDBKey(db_key.c_str())
{
}
inline gm::string&
DynamicGrammarMessage::GetDBKey(void)
{
        return cDBKey;
}
#endif
```

```
/*
 * DynamicGrammarPhraseMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:16 $
 */
#ifndef DYNAMICGRAMMARPHRASEMSG_H
#define DYNAMICGRAMMARPHRASEMSG_H
#include "DynamicGrammarMsg.h"
class __declspec(dllexport) DynamicGrammarPhraseMessage : public DynamicGrammarMessage
{
        gm::string cPhraseID;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(DynamicGrammarPhraseMessage, DynamicGrammarMessage);
        inline DynamicGrammarPhraseMessage(unsigned short id, CallToken& token,
                const gm::string& db_key, const gm::string &phrase_id);
        inline gm::string& GetPhraseID(void);
};
inline
DynamicGrammarPhraseMessage::DynamicGrammarPhraseMessage(unsigned short id,
        CallToken& token, const gm::string& db_key, const gm::string &phrase_id)
        : DynamicGrammarMessage(id, token, db_key), cPhraseID(phrase_id.c_str())
{
}
inline gm::string&
DynamicGrammarPhraseMessage::GetPhraseID(void)
{
        return cPhraseID;
}
#endif
```

```
/*
 * DynamicGrammarPhraseResultNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:16 $
 */
#ifndef DYNAMICGRAMMARPHRASERESULTNTFY_H
#define DYNAMICGRAMMARPHRASERESULTNTFY_H
#include "DynamicGrammarResultNtfy.h"
class __declspec(dllexport) DynamicGrammarPhraseResultNotification : public DynamicGrammarResultNotification
{
        gm::string cPhraseID;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(DynamicGrammarPhraseResultNotification,
DynamicGrammarResultNotification);
        inline DynamicGrammarPhraseResultNotification(unsigned short cmdid, CallToken& ct,
                const gm::string& db_key, Result result, const gm::string& phrase_id);
        inline gm::string& GetPhraseID(void);
};
inline
DynamicGrammarPhraseResultNotification::DynamicGrammarPhraseResultNotification(unsigned short cmdid,
        CallToken& callToken, const gm::string& db_key, DynamicGrammarResultNotification::Result result,
        const gm::string& phrase_id)
        : DynamicGrammarResultNotification(cmdid, callToken, db_key, result), cPhraseID(phrase_id.c_str())
{
}
inline gm::string&
DynamicGrammarPhraseResultNotification::GetPhraseID(void)
{
        return cPhraseID;
}
#endif
```

```
/*
 * DynamicGrammarResultNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:21:16 $
 */
#ifndef DYNAMICGRAMMARRESULTNTFY_H
#define DYNAMICGRAMMARRESULTNTFY_H
#include "OneCallTokenMsgBase.h"
class __declspec(dllexport) DynamicGrammarResultNotification : public OneCallTokenMessageBase
{
public:
        typedef enum
        {
                R_Success,
                R_CantOpenDatabase,
                R_CantFindDynamicGrammar,
                R_CantFindParentGrammar,
                R_Failure
        } Result;
private:
        gm::string cDBKey;
        Result cResult;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(DynamicGrammarResultNotification,
OneCallTokenMessageBase);
        inline DynamicGrammarResultNotification(unsigned short cmdid, CallToken& ct,
                const gm::string& db_key, Result result);
        inline gm::string& GetDatabaseKey(void);
        inline Result GetResult(void);
};
inline
DynamicGrammarResultNotification::DynamicGrammarResultNotification(unsigned short cmdid,
        CallToken& callToken, const gm::string& db_key, DynamicGrammarResultNotification::Result result)
        : OneCallTokenMessageBase(cmdid, callToken), cDBKey(db_key.c_str()), cResult(result)
{
}
inline gm::string&
DynamicGrammarResultNotification::GetDatabaseKey(void)
{
        return cDBKey;
}
inline DynamicGrammarResultNotification::Result
DynamicGrammarResultNotification::GetResult(void)
{
        return cResult;
}
```

#endif

Document provided by:

```
/*
 * EndCompositePromptMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:17 $
 */
#ifndef ENDCOMPOSITEPROMPTMSG_H
#define ENDCOMPOSITEPROMPTMSG_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) EndCompositePromptMessage
        : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(EndCompositePromptMessage,
                OneCallTokenMessageBase);
        inline EndCompositePromptMessage(CallToken& token);
};
inline
EndCompositePromptMessage::EndCompositePromptMessage(
        CallToken& token)
        : OneCallTokenMessageBase(M_CMD_ENDCOMPOSITEPROMPT, token)
{
}
#endif
```

```
/*
 * EndSpeechNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:17 $
 */
#ifndef ENDSPEECHNTFY_H
#define ENDSPEECHNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) EndSpeechNotification : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(EndSpeechNotification,
                OneCallTokenMessageBase);
        inline EndSpeechNotification(CallToken& token);
};
inline
EndSpeechNotification::EndSpeechNotification(CallToken& token)
        : OneCallTokenMessageBase(M_NTFY_ENDSPEECH, token)
{
}
#endif
```

```
/*
 * FailureNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.6 $
 * $Date: 2000/12/07 03:21:17 $
 */
#ifndef FAILURENTFY_H
#define FAILURENTFY_H
#include "OneCallTokenMsgBase.h"
class __declspec(dllexport) FailureNotification : public OneCallTokenMessageBase
{
        SerialNumberT cFailureSerialNo;
        unsigned long cInfo;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(FailureNotification, OneCallTokenMessageBase);
        inline FailureNotification(CallToken& ct, unsigned short cmdid,
                SerialNumberT FailureSerialNo, unsigned long Info);
        inline SerialNumberT GetFailureSerialNo(void);
        inline unsigned long GetInfo(void);
};
inline
FailureNotification::FailureNotification(CallToken& callToken, unsigned short cmdid,
        SerialNumberT FailureSerialNo, unsigned long Info)
        : OneCallTokenMessageBase(cmdid, callToken)
{
        cFailureSerialNo = FailureSerialNo;
        cInfo = Info;
}
inline CMSMessage::SerialNumberT
FailureNotification::GetFailureSerialNo(void)
{
        return cFailureSerialNo;
}
inline unsigned long
FailureNotification::GetInfo(void)
{
        return cInfo;
}
#endif
```

```
/*
 * GetPropertyMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:17 $
 */
#ifndef GETPROPERTYMSG_H
#define GETPROPERTYMSG_H
#include "PropertyMsg.h"
#include "VPPDefs.h"
class __declspec(dllexport) GetPropertyMessage : public PropertyMessage
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(GetPropertyMessage, PropertyMessage);
        inline GetPropertyMessage(const gm::string& name);
};
inline
GetPropertyMessage::GetPropertyMessage(const gm::string& name)
        : PropertyMessage(M_CMD_GETPROPERTY, name)
{
}
#endif
```

```
/*
 * GlobalCallTChannel.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.23 $
 * $Date: 2000/12/07 03:28:21 $
 */
#ifndef GLOBALCALLTCHANNEL_H
#define GLOBALCALLTCHANNEL_H
#include "MTSLib/TelephonyChannel.h"
#include "GlobalCallTSP.h"
#include "VPPUtils/CriticalSection.h"
#include <gclib.h>
#include <stdarg.h>
class __declspec(dllexport) GlobalCallTChannel : public TelephonyChannel
{
public:
        typedef enum {
                // Common States
                                                        // ---------------------------------------------
                eNullState,                     // No call is assigned to the device
                                                        // (time slot or line); call released.
                                                        // ---------------------------------------------
                eConnectedState, // A connection is established for an inbound or
                                                        // outbound call.  Call charge begins.
                                                        // ---------------------------------------------
                eDisconnectedState,     // Network diconnects the call.  Subsequently,
                                                        // the application drops the call and releases
                                                        // the CRN and other resources used for call.
                                                        // ---------------------------------------------
                eIdleState,                     // The call was dropped; call is not active.
                                                        // Subsequently the application releases the CRN
                                                        // and other resources used for the call.
                                                        // ---------------------------------------------
                // Incoming only states
                                                        // ---------------------------------------------
                eOfferedState,          // An inbound call from the network is offered
                                                        // to the application; call received.
                                                        // ---------------------------------------------
                eAcceptedState,         // Indicates an acknowledgment, such as ringing,
                                                        // ringback, etc., to the calling party, that an
                                                        // inbound call is received but not yet connected
                                                        // to the called party; call accepted.
                                                        // ---------------------------------------------
                // Outgoing only states
                                                        // ---------------------------------------------
                eDialingState,          // Call establishment is in progress; outbound
                                                        // call request. Dialing information was sent to,
                                                        // and acknowledged, by the network.
                                                        // ---------------------------------------------
                eAlertingState          // The destination was reached and the application
```

```
                                        // is waiting for the destination party to answer
                                        // the call; call alerted sent or received. This
                                        // state event may be reported to the application
                                        // or may be masked.
                                        // ---------------------------------------------
        } MyCallStates;
        GlobalCallTChannel(ServiceProvider *pServiceProvider, int boardNum, int portNum, const char *logType,
int id);
        virtual ~GlobalCallTChannel(void);
        bool Setup(const gm::string& chname);
        bool Shutdown(void);
        static long dt_handler(unsigned long evhandle);
        static bool EnableHandler(void);
        inline long GetHdl(void);
        inline bool IsISDN(void);
        bool IsAlive(void);
        virtual void NotifyBoundToSession(SessionNode *pSessionNode);
        virtual bool IsAvailable(void);
  bool SetPhysicalChannelStatus(TelephonyChannel::ChannelStatus status);
protected:
        int LoadMetaEventData(void);
        virtual bool WaitForIncomingCall(void);
        //
        // From TelephonyChannel
        //
        virtual unsigned long PlaceCall(const gm::string& calledNumber,
                const gm::string& callingNumber, unsigned char *uui = NULL, int uuiLength = 0);
        virtual unsigned long CancelRequest(void);
        virtual unsigned long DropCall(void);
        virtual unsigned long ReleaseCall(void);
        virtual unsigned long ListenTo(unsigned short timeslot);
        virtual unsigned long PlaceCallOnHold(void);
        virtual unsigned long AcceptCall(void);
        virtual unsigned long RejectCall(TelephonyChannel::RejectCause cause);
        friend class GlobalCallTSP;
private:
        LINEDEV                 cHdl; // LINEDEV is long
        int                     cBoardNum;
        int                     cPortNum;
        bool                    cWaitingForCall;
        bool                    cReady;
        gm::string              cChanInfo;
        CRN                     cCrnData;
        METAEVENT               cMetaevent;
        char                    cANIBuf [GC_ADDRSIZE];
        char                    cDNISBuf[GC_ADDRSIZE];
        MyCallStates    cMyCallState;
        static CriticalSection cCriticalSection;
        static int              cSequenceNumber;
        unsigned short  cFlags;
        bool                    cIncomingCallValid;
        bool                    cNotifyCallDropped;
        HANDLE                  cInCallConnectedEvt;
        HANDLE                  cWaitingForTerminationAckEvt;
        unsigned long   cTimeChannelReleased;
        enum {
```

```
                F_ISDN = 0x1
        };
        static char const *cStates[9];
        inline void SetIsISDN(bool flag);
        inline TelephonyServiceProvider *GetTSP(void);
        bool HandleEvent(unsigned long evhandle);
        //static long dt_handler(unsigned long evhandle);
        virtual bool handleNullEvents(int evhandle);
        virtual bool handleConnectedEvents(int evhandle);
        virtual bool handleDisconnectedEvents(int evhandle);
        virtual bool handleIdleEvents(int evhandle);
        virtual bool handleOfferedEvents(int evhandle);
        virtual bool handleAcceptedEvents(int evhandle);
        virtual bool handleDialingEvents(int evhandle);
        virtual bool handleAlertingEvents(int evhandle);

        virtual bool acceptInboundCall(void);
        virtual unsigned long disconnectCall(int cause);
        void AssignSequenceNumber(void);
        bool GetBChannelStatus(int *status);
        bool GetDChannelStatus(int *status);
        void SetChannelState(void);
        void GetMyCallStateString(gm::string& myCallStateStr);
        const char const *GetCallState(void);
        virtual const char *GetIDString(void);
        void CheckErrorUsingResultValue(const gm::string& messageHeader);
        void CheckError(int retCode, const char const *funcName);
        const gm::string& GetChanInfoStr(void);
        void printMessage(const char *group, const char *fmt, ...);
};
inline void
GlobalCallTChannel::SetIsISDN(bool flag)
{
        if (flag)
                cFlags |= F_ISDN;
        else
                cFlags &= ~F_ISDN;
}
inline bool
GlobalCallTChannel::IsISDN(void)
{
        return (bool)((cFlags & F_ISDN) != 0);
}
inline TelephonyServiceProvider *
GlobalCallTChannel::GetTSP(void)
{
        return (GlobalCallTSP *)cServiceProvider;
}
inline long
GlobalCallTChannel::GetHdl(void)
{
        return cHdl;
}
#endif
```

```
/*
 * GlobalCallTSP.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.10 $
 * $Date: 2000/12/07 03:28:50 $
 */
#ifndef GLOBALCALLTSP_H
#define GLOBALCALLTSP_H
#include "MTSLib/TelephonyServiceProvider.h"
#include <windows.h>
#include <process.h>
#include "VPPUtils/Log.h"
class GlobalCallTChannel;
class __declspec(dllexport) GlobalCallTSP : public TelephonyServiceProvider
{
public:
        static const char const *protocolName;
        GlobalCallTSP(MTSServer *pServer);
        virtual ~GlobalCallTSP(void);
        virtual bool Setup(void);
        virtual bool Teardown(void);
   virtual bool BeginService(void);
        virtual bool Restart(void);
        void BringAllChannelsDown(void);
        GlobalCallTChannel *FindChannelForDescriptor(long desc);
        static GlobalCallTSP *GetInstance(void);
        static void SetInstance(GlobalCallTSP *ptr);
//protected:
        //friend class GlobalCallTChannel;
        //friend class DialogicMChannel;
        //bool MonitorDevice(long deviceHandle);
private:
        static GlobalCallTSP *cInstance;
        static const char const *cLogType;
//      long* pDeviceHandles;
//      int deviceCount;
//      int deviceArrSize;
//      CriticalSection cCritSection;
        HANDLE      cStopTSPEvent; // Signaled when Teardown is called
        HANDLE      cThreadH;
        DWORD       cThreadId;
        static unsigned __stdcall _GoThreadFunc(LPVOID);
        unsigned GoThreadFunc(void);
        bool go();

        bool NewChannel(int boardNum, int portNum, int id);
};
class GCTSPLog : public LogT<GCTSPLog> {
   GCTSPLog() : LogT<GCTSPLog>("GLCL_TSP") {};
   static GCTSPLog log; // for initialization
```

```
};
#endif
```

Document #: 2000000

```
/*
 * GMLangID.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:22:22 $
 */
#ifndef GMLANGID_H
#define GMLANGID_H
typedef unsigned short GMLANGID;
#define MAKEGMLANGID(p, s)          ((((GMLANGID)(s)) << 10) | (GMLANGID)(p))
#define GMPRIMARYLANGID(lgid)       ((GMLANGID)(lgid) & 0x3ff)
#define GMSUBLANGID(lgid)           ((GMLANGID)(lgid) >> 10)
//
// Primary language IDs.
//
#define GMLANG_AFRIKAANS             0x36
#define GMLANG_ALBANIAN              0x1c
#define GMLANG_ARABIC               0x01
#define GMLANG_BASQUE               0x2d
#define GMLANG_BELARUSIAN             0x23
#define GMLANG_BULGARIAN              0x02
#define GMLANG_CATALAN              0x03
#define GMLANG_CHINESE              0x04
#define GMLANG_CROATIAN              0x1a
#define GMLANG_CZECH               0x05
#define GMLANG_DANISH               0x06
#define GMLANG_DUTCH               0x13
#define GMLANG_ENGLISH              0x09
#define GMLANG_ESTONIAN              0x25
#define GMLANG_FAEROESE              0x38
#define GMLANG_FARSI               0x29
#define GMLANG_FINNISH              0x0b
#define GMLANG_FRENCH               0x0c
#define GMLANG_GERMAN               0x07
#define GMLANG_GREEK               0x08
#define GMLANG_HEBREW               0x0d
#define GMLANG_HUNGARIAN              0x0e
#define GMLANG_ICELANDIC              0x0f
#define GMLANG_INDONESIAN             0x21
#define GMLANG_ITALIAN              0x10
#define GMLANG_JAPANESE              0x11
#define GMLANG_KOREAN               0x12
#define GMLANG_LATVIAN              0x26
#define GMLANG_LITHUANIAN             0x27
#define GMLANG_NORWEGIAN              0x14
#define GMLANG_POLISH               0x15
#define GMLANG_PORTUGUESE             0x16
#define GMLANG_ROMANIAN              0x18
#define GMLANG_RUSSIAN              0x19
#define GMLANG_SERBIAN              0x1a
```

```
#define GMLANG_SLOVAK               0x1b
#define GMLANG_SLOVENIAN            0x24
#define GMLANG_SPANISH              0x0a
#define GMLANG_SWEDISH              0x1d
#define GMLANG_THAI                 0x1e
#define GMLANG_TURKISH              0x1f
#define GMLANG_UKRAINIAN            0x22
#define GMLANG_VIETNAMESE           0x2a
#define GMLANG_INVALID              0x3ff
//
// Sublanguage IDs.
//
// The name immediately following GMSUBLANG_ dictates which primary
// language ID that sublanguage ID can be combined with to form a
// valid language ID.
//
#define GMSUBLANG_NONE                  0x00   // For languages with no sublang
#define GMSUBLANG_ARABIC_SAUDI_ARABIA   0x01   // Arabic (Saudi Arabia)
#define GMSUBLANG_ARABIC_IRAQ           0x02   // Arabic (Iraq)
#define GMSUBLANG_ARABIC_EGYPT          0x03   // Arabic (Egypt)
#define GMSUBLANG_ARABIC_LIBYA          0x04   // Arabic (Libya)
#define GMSUBLANG_ARABIC_ALGERIA        0x05   // Arabic (Algeria)
#define GMSUBLANG_ARABIC_MOROCCO        0x06   // Arabic (Morocco)
#define GMSUBLANG_ARABIC_TUNISIA        0x07   // Arabic (Tunisia)
#define GMSUBLANG_ARABIC_OMAN           0x08   // Arabic (Oman)
#define GMSUBLANG_ARABIC_YEMEN          0x09   // Arabic (Yemen)
#define GMSUBLANG_ARABIC_SYRIA          0x0a   // Arabic (Syria)
#define GMSUBLANG_ARABIC_JORDAN         0x0b   // Arabic (Jordan)
#define GMSUBLANG_ARABIC_LEBANON        0x0c   // Arabic (Lebanon)
#define GMSUBLANG_ARABIC_KUWAIT         0x0d   // Arabic (Kuwait)
#define GMSUBLANG_ARABIC_UAE            0x0e   // Arabic (U.A.E)
#define GMSUBLANG_ARABIC_BAHRAIN        0x0f   // Arabic (Bahrain)
#define GMSUBLANG_ARABIC_QATAR          0x10   // Arabic (Qatar)
#define GMSUBLANG_CHINESE_TRADITIONAL   0x01   // Chinese (Taiwan)
#define GMSUBLANG_CHINESE_SIMPLIFIED    0x02   // Chinese (PR China)
#define GMSUBLANG_CHINESE_HONGKONG      0x03   // Chinese (Hong Kong)
#define GMSUBLANG_CHINESE_SINGAPORE     0x04   // Chinese (Singapore)
#define GMSUBLANG_DUTCH_NETHERLANDS     0x01   // Dutch (Netherlands)
#define GMSUBLANG_DUTCH_BELGIAN         0x02   // Dutch (Belgian)
#define GMSUBLANG_ENGLISH_US            0x01   // English (USA)
#define GMSUBLANG_ENGLISH_UK            0x02   // English (UK)
#define GMSUBLANG_ENGLISH_AUS           0x03   // English (Australian)
#define GMSUBLANG_ENGLISH_CAN           0x04   // English (Canadian)
#define GMSUBLANG_ENGLISH_NZ            0x05   // English (New Zealand)
#define GMSUBLANG_ENGLISH_EIRE          0x06   // English (Irish)
#define GMSUBLANG_ENGLISH_SOUTH_AFRICA  0x07   // English (South Africa)
#define GMSUBLANG_ENGLISH_JAMAICA       0x08   // English (Jamaica)
#define GMSUBLANG_ENGLISH_CARIBBEAN     0x09   // English (Caribbean)
#define GMSUBLANG_ENGLISH_BELIZE        0x0a   // English (Belize)
#define GMSUBLANG_ENGLISH_TRINIDAD      0x0b   // English (Trinidad)
#define GMSUBLANG_FRENCH_FRANCE         0x01   // French (France)
#define GMSUBLANG_FRENCH_BELGIAN        0x02   // French (Belgian)
#define GMSUBLANG_FRENCH_CANADIAN       0x03   // French (Canadian)
#define GMSUBLANG_FRENCH_SWISS          0x04   // French (Swiss)
#define GMSUBLANG_FRENCH_LUXEMBOURG     0x05   // French (Luxembourg)
#define GMSUBLANG_GERMAN_GERMANY        0x01   // German (Germany)
```

```
#define GMSUBLANG_GERMAN_SWISS          0x02   // German (Swiss)
#define GMSUBLANG_GERMAN_AUSTRIAN       0x03   // German (Austrian)
#define GMSUBLANG_GERMAN_LUXEMBOURG     0x04   // German (Luxembourg)
#define GMSUBLANG_GERMAN_LIECHTENSTEIN  0x05   // German (Liechtenstein)
#define GMSUBLANG_ITALIAN_ITALY         0x01   // Italian (Italy)
#define GMSUBLANG_ITALIAN_SWISS         0x02   // Italian (Swiss)
#define GMSUBLANG_KOREAN_KOREA          0x01   // Korean (Extended Wansung)
#define GMSUBLANG_KOREAN_JOHAB          0x02   // Korean (Johab)
#define GMSUBLANG_NORWEGIAN_BOKMAL      0x01   // Norwegian (Bokmal)
#define GMSUBLANG_NORWEGIAN_NYNORSK     0x02   // Norwegian (Nynorsk)
#define GMSUBLANG_PORTUGUESE_PORTUGAL   0x02   // Portuguese (Portugal)
#define GMSUBLANG_PORTUGUESE_BRAZILIAN  0x01   // Portuguese (Brazilian)
#define GMSUBLANG_SERBIAN_LATIN         0x02   // Serbian (Latin)
#define GMSUBLANG_SERBIAN_CYRILLIC      0x03   // Serbian (Cyrillic)
#define GMSUBLANG_SPANISH_SPAIN         0x01   // Spanish (Castilian)
#define GMSUBLANG_SPANISH_MEXICAN       0x02   // Spanish (Mexican)
#define GMSUBLANG_SPANISH_MODERN        0x03   // Spanish (Modern)
#define GMSUBLANG_SPANISH_GUATEMALA     0x04   // Spanish (Guatemala)
#define GMSUBLANG_SPANISH_COSTA_RICA    0x05   // Spanish (Costa Rica)
#define GMSUBLANG_SPANISH_PANAMA        0x06   // Spanish (Panama)
#define GMSUBLANG_SPANISH_DOMINICAN_REPUBLIC 0x07 // Spanish (Dominican Republic)
#define GMSUBLANG_SPANISH_VENEZUELA     0x08   // Spanish (Venezuela)
#define GMSUBLANG_SPANISH_COLOMBIA      0x09   // Spanish (Colombia)
#define GMSUBLANG_SPANISH_PERU          0x0a   // Spanish (Peru)
#define GMSUBLANG_SPANISH_ARGENTINA     0x0b   // Spanish (Argentina)
#define GMSUBLANG_SPANISH_ECUADOR       0x0c   // Spanish (Ecuador)
#define GMSUBLANG_SPANISH_CHILE         0x0d   // Spanish (Chile)
#define GMSUBLANG_SPANISH_URUGUAY       0x0e   // Spanish (Uruguay)
#define GMSUBLANG_SPANISH_PARAGUAY      0x0f   // Spanish (Paraguay)
#define GMSUBLANG_SPANISH_BOLIVIA       0x10   // Spanish (Bolivia)
#define GMSUBLANG_SPANISH_EL_SALVADOR   0x11   // Spanish (El Salvador)
#define GMSUBLANG_SPANISH_HONDURAS      0x12   // Spanish (Honduras)
#define GMSUBLANG_SPANISH_NICARAGUA     0x13   // Spanish (Nicaragua)
#define GMSUBLANG_SPANISH_PUERTO_RICO   0x14   // Spanish (Puerto Rico)
#define GMSUBLANG_SWEDISH_SWEEDEN       0x01   // Swedish (Sweeden)
#define GMSUBLANG_SWEDISH_FINLAND       0x02   // Swedish (Finland)
#define GMSUBLANG_INVALID               0x3f
#define INVALID_GMLANGID MAKEGMLANGID(GMLANG_INVALID, GMSUBLANG_INVALID)
#endif
```

```
/*
 * GMLangMgr.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:22:23 $
 */
#ifndef GMLANGMGR_H
#define GMLANGMGR_H
#include "GMString"
#include "GMLangID.h"
class __declspec(dllexport) GMLanguageManager
{
public:
        static bool GMLANGIDToStrings(GMLANGID langid, gm::string& primary,
                gm::string& sublang);
        static GMLANGID StringsToGMLANGID(const gm::string& primary,
                const gm::string& sublang);
};
#endif
```

Document Produced by Delphi

```
/*
 * GrammarHandle.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:20:06 $
 */
#ifndef _GRAMMAR_HANDLE_H
#define _GRAMMAR_HANDLE_H
#include <VPPUtils/GMString>
class __declspec(dllexport) GrammarHandle {
public:
   enum HandleType {
      DYNAMIC,
      BUILTIN
   } cHandleType;
   gm::string key;
   GrammarHandle() : cHandleType(BUILTIN), key() {
      key = "";
   }
   GrammarHandle(GrammarHandle& handle) :
      cHandleType(handle.cHandleType), key() {
      key = handle.key.c_str();
   }
   GrammarHandle(HandleType type, gm::string& k) :
      cHandleType(type) {
      key = k.c_str();
   }
};
#endif
```

```
/*
 * HandoffRecordingMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:18 $
 */
#ifndef HANDOFFRECORDINGMSG_H
#define HANDOFFRECORDINGMSG_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
#include "VPPUtils/GMString"
class __declspec(dllexport) HandOffRecordingMessage
        : public OneCallTokenMessageBase
{
        RECORDINGID cRecordingID;
        gm::string cServiceName;
        unsigned char *cData;
        unsigned long cDataLength;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);

public:
        DECLARE_CMSMESSAGE_SUBCLASS(HandOffRecordingMessage,
                OneCallTokenMessageBase);
        HandOffRecordingMessage(CallToken& token, RECORDINGID RecordingID,
                const gm::string& Servicename, void *pData, unsigned long DataLength);
        ~HandOffRecordingMessage(void);
        inline RECORDINGID GetRecordingID(void);
        inline const gm::string& GetServiceName(void);
        inline void *GetData(void);
        inline unsigned long GetDataLength(void);
};
inline RECORDINGID
HandOffRecordingMessage::GetRecordingID(void)
{
        return cRecordingID;
}
inline const gm::string&
HandOffRecordingMessage::GetServiceName(void)
{
        return cServiceName;
}
inline void *
HandOffRecordingMessage::GetData(void)
{
        return cData;
}
inline unsigned long
HandOffRecordingMessage::GetDataLength(void)
```

```
{
        return cDataLength;
}
#endif
```

Document: 0B0B2B5D

```
/*
 * Hashtable.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.13 $
 * $Date: 2000/12/07 03:22:23 $
 */
#ifndef _HASHTABLE_H
#define _HASHTABLE_H
#include "Monitor.h"
#include "Synchronized.h"
template<class Key, class Value>
class __declspec(dllexport) Hashtable : public Monitor {
  static Key NULL_KEY;
  static Value NULL_VALUE;
  class HashtableEntry {
  public:
    int hash;
    Key key;
    Value value;
    HashtableEntry* next;
    HashtableEntry* clone() {
      HashtableEntry* entry = new HashtableEntry();
      entry->hash = hash;
      entry->key = key;
      entry->value = value;
      entry->next = (next != NULL) ? next->clone() : NULL;
      return entry;
    }
    void deleteKeys(Hashtable<Key,Value>* table) {
      if (key != NULL_KEY) {
        table->_deleteKey(key);
        key = NULL_KEY;
      }
      if (next != NULL) {
        next->deleteKeys(table);
      }
    }
    ~HashtableEntry() {
      // delete key;
      if (next != NULL) {
        delete next;
        next = NULL;
      }
    }
  };
  typedef HashtableEntry* HashtableEntry_ptr;
public:
  class __declspec(dllexport) Iterator {
  private:
    HashtableEntry_ptr* _table;
```

```
    HashtableEntry_ptr current;
    int table_size;
    int index;
  public:
    Iterator(HashtableEntry_ptr* t, int s) : _table(t), table_size(s), index(0), current(NULL) {
    };
    ~Iterator() {
    };
    bool next() {
      while (index < table_size) {
        if (current == NULL) {
          current = _table[index];
        } else {
          current = current->next;
        }
        if (current != NULL) {
          return TRUE;
        }
        index ++;
      }
      return FALSE;
    };
    Key key() {
      return current->key;
    };
    Value value() {
      return current->value;
    };
  };
  Hashtable(int initialCapacity, float lf) : table(NULL), table_size(initialCapacity), loadFactor(lf), count(0),
threshold(0) {
    if ((initialCapacity <= 0) || (loadFactor <= 0.0)) {
      //throw new IllegalArgumentException();
    }
    threshold = (int)(table_size * loadFactor);
    createTable();
  }
  Hashtable() : table(NULL), table_size(101), loadFactor(0.75f), count(0), threshold((int)(table_size * loadFactor)) {
    threshold = (int)(table_size * loadFactor);
    createTable();
  }
  virtual ~Hashtable() {
    Synchronized sync(this);
    HashtableEntry_ptr* tab = table;
    for (int index = table_size; --index >= 0; ) {
      if (tab[index] != NULL) {
        // hashtable needs to be empty when deleted
        // tab[index]->deleteKeys(this);
        delete tab[index];
      }
      tab[index] = NULL;
    }
    count = 0;
    delete table;
  }
  void createTable() {
```

```
    table = new HashtableEntry_ptr[table_size];
    for (int i=0; i<table_size; i++) {
      table[i] = NULL;
    }
  }
  /**
   */
  int size() {
    return count;
  }
  bool isEmpty() {
    return count == 0;
  }
  bool contains(Value value) {
    Synchronized sync(this);
    if (value == NULL_VALUE) {
      return FALSE;
    }
    HashtableEntry_ptr* tab = table;
    for (int i = table_size ; i-- > 0 ;) {
      for (HashtableEntry* e = tab[i] ; e != NULL ; e = e->next) {
        if (_valueEquals(e->value, value)) {
          return true;
        }
      }
    }
    return false;
  }
  bool containsKey(Key key) {
    Synchronized sync(this);
    HashtableEntry_ptr* tab = table;
    int hash = _hash(key);
    int index = (hash & 0x7FFFFFFF) % table_size;
    for (HashtableEntry_ptr e = tab[index] ; e != NULL ; e = e->next) {
      int h = _hash(e->key);
      if ((h == hash) && _keyEquals(e->key, key)) {
        return true;
      }
    }
    return false;
  }
  Value get(Key key) {
    Synchronized sync(this);
    HashtableEntry_ptr* tab = table;
    int hash = _hash(key);
    int index = (hash & 0x7FFFFFFF) % table_size;
    for (HashtableEntry_ptr e = tab[index] ; e != NULL ; e = e->next) {
      if ((e->hash == hash) && _keyEquals(e->key, key)) {
        return e->value;
      }
    }
    return NULL_VALUE;
  }
  Value put(Key key, Value value) {
    Synchronized sync(this);
    // Make sure the value is not null
```

```
    if (value == NULL_VALUE) {
      return NULL_VALUE;
      // throw new NullPointerException();
    }
    // Makes sure the key is not already in the hashtable.
    HashtableEntry_ptr* tab = table;
    int hash = _hash(key);
    int index = (hash & 0x7FFFFFFF) % table_size;
    for (HashtableEntry_ptr e = tab[index] ; e != NULL ; e = e->next) {
      if ((e->hash == hash) && _keyEquals(e->key, key)) {
        Value old = e->value;
        e->value = value;
        return old;
      }
    }
    if (count >= threshold) {
      // Rehash the table if the threshold is exceeded
      rehash();
      return put(key, value);
    }
    // Creates the new entry.
    HashtableEntry* new_e = new HashtableEntry();
    new_e->hash = hash;
    new_e->key = _copy(key);
    new_e->value = value;
    new_e->next = tab[index];
    tab[index] = new_e;
    count++;
    return NULL_VALUE;
  }
  Value remove(Key key) {
    HashtableEntry_ptr* tab = table;
    int hash = _hash(key);
    int index = (hash & 0x7FFFFFFF) % table_size;
    for (HashtableEntry_ptr e = tab[index], prev = NULL ;
        e != NULL ;
        prev = e, e = e->next) {
      if ((e->hash == hash) && _keyEquals(e->key, key)) {
        if (prev != NULL) {
          prev->next = e->next;
        } else {
          tab[index] = e->next;
        }
        count--;
        Value value = e->value;
        e->next = NULL;
        e->deleteKeys(this);
        delete e;
        return value;
      }
    }
    return NULL_VALUE;
  }
  /**
   * Clears this hashtable so that it contains no keys.
   */
```

```
    void clear() {
        Synchronized sync(this);
        HashtableEntry_ptr* tab = table;
        for (int index = table_size; --index >= 0; ) {
            if (tab[index] != NULL) {
                tab[index]->deleteKeys(this);
                delete tab[index];
            }
            tab[index] = NULL;
        }
        count = 0;
    }
    Iterator iterator() {
        return Iterator(table, table_size);
    };
protected:
    friend class HashtableEntry;
    virtual int _hash(Key k) = 0;
    virtual Key _copy(Key k) = 0;
    virtual bool _valueEquals(Value v1, Value v2) = 0;
    virtual bool _keyEquals(Key k1, Key k2) = 0;
    virtual void _deleteKey(Key key) = 0;
    // virtual Value _null() = 0;
    void rehash() {
        int oldCapacity = table_size;
        HashtableEntry_ptr* oldTable = table;
        table_size = oldCapacity * 2 + 1;
        createTable();
        threshold = (int)(table_size * loadFactor);
        for (int i = oldCapacity ; i-- > 0 ;) {
            for (HashtableEntry* old = oldTable[i] ; old != NULL ; ) {
                HashtableEntry* e = old;
                old = old->next;
                int index = (e->hash & 0x7FFFFFFF) % table_size;
                e->next = table[index];
                table[index] = e;
            }
        }
        delete oldTable;
    };
private:
    /**
     * The hash table data.
     */
    HashtableEntry_ptr* table;
    int table_size;
    /**
     * The total number of entries in the hash table.
     */
    int count;
    /**
     * The load factor for the hashtable.
     */
    float loadFactor;
    /**
     * Rehashes the table when count exceeds this threshold.
```

```
    */
    int threshold;
};
#endif
```

Downloaded from http://www.everyspec.com

```
/*
 * IDLog.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:22:23 $
 */
#ifndef  _IDLOG_H
#define  _IDLOG_H
#include "VPPUtils\Log.h"
class __declspec(dllexport) IDLog {
public:
   IDLog(const char* name);
   virtual void setLogID(const char* string);
   virtual void setLogID(const char* fmt, ...);
   void error(const char* fmt, ...);
   void warning(const char* fmt, ...);
   void info(const char* fmt, ...);
   void diag(const char* fmt, ...);
   void debug(const char* fmt, ...);
private:
   virtual void va_setLogID(const char* fmt, const va_list vl);
   VPPLog* log;
   gm::string mID;                              // Additional log info (channel ID, session ID, ...)
};
#endif
```

Confidential - Attorneys Eyes Only

```
/*
 * IncomingCallNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:18 $
 */
#ifndef INCOMINGCALLNTFY_H
#define INCOMINGCALLNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
#include "Address.h"
class __declspec(dllexport) IncomingCallNotification : public OneCallTokenMessageBase
{
        Address cDNIS;
        Address cANI;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);

public:
        DECLARE_CMSMESSAGE_SUBCLASS(IncomingCallNotification, OneCallTokenMessageBase);
        inline IncomingCallNotification(CallToken& ct, Address& DNIS, Address& ANI);
        inline Address& GetDNIS(void);
        inline Address& GetANI(void);
};
inline
IncomingCallNotification::IncomingCallNotification(CallToken& ct, Address& DNIS,
        Address& ANI) : OneCallTokenMessageBase(M_NTFY_INCOMINGCALL, ct)
{
        cDNIS = DNIS;
        cANI = ANI;
}
inline Address&
IncomingCallNotification::GetDNIS(void)
{
        return cDNIS;
}
inline Address&
IncomingCallNotification::GetANI(void)
{
        return cANI;
}
#endif
```

```
/*
 * IncomingCallRevokedNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:18 $
 */
#ifndef INCOMINGCALLREVOKEDNTFY_H
#define INCOMINGCALLREVOKEDNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) IncomingCallRevokedNotification : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(IncomingCallRevokedNotification, OneCallTokenMessageBase);
        inline IncomingCallRevokedNotification(CallToken& ct);
};
inline
IncomingCallRevokedNotification::IncomingCallRevokedNotification(CallToken& ct)
        : OneCallTokenMessageBase(M_NTFY_INCOMINGCALLREVOKED, ct)
{
}
#endif
```

Confidential - Subject to Protective Order

```
/*
 * InsertDynamicGrammarMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:19 $
 */
#ifndef INSERTDYNAMICGRAMMARMSG_H
#define INSERTDYNAMICGRAMMARMSG_H
#include "DynamicGrammarMsg.h"
class __declspec(dllexport) InsertDynamicGrammarMessage : public DynamicGrammarMessage
{
        gm::string cGrammarName;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(InsertDynamicGrammarMessage, DynamicGrammarMessage);
        inline InsertDynamicGrammarMessage(CallToken& token, const gm::string& db_key,
                const gm::string &grammar_name);
        inline gm::string& GetGrammarName(void);
};
inline
InsertDynamicGrammarMessage::InsertDynamicGrammarMessage(CallToken& token,
        const gm::string& db_key, const gm::string& grammar_name)
        : DynamicGrammarMessage(M_CMD_INSERTDYNAMICGRAMMAR, token, db_key),
        cGrammarName(grammar_name.c_str())
{
}
inline gm::string&
InsertDynamicGrammarMessage::GetGrammarName(void)
{
        return cGrammarName;
}
#endif
```

```
/*
 * InsertDynamicGrammarResultNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:19 $
 */
#ifndef INSERTDYNAMICGRAMMARRESULTNTFY_H
#define INSERTDYNAMICGRAMMARRESULTNTFY_H
#include "VPPDefs.h"
#include "DynamicGrammarResultNtfy.h"
class __declspec(dllexport) InsertDynamicGrammarResultNotification : public DynamicGrammarResultNotification
{
        gm::string cGrammarName;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(InsertDynamicGrammarResultNotification,
DynamicGrammarResultNotification);
        inline InsertDynamicGrammarResultNotification(CallToken& ct, const gm::string& db_key, Result result,
                const gm::string& grammar_name);
        inline gm::string& GetGrammarName(void);
};
inline
InsertDynamicGrammarResultNotification::InsertDynamicGrammarResultNotification(CallToken& callToken,
        const gm::string& db_key, DynamicGrammarResultNotification::Result result, const gm::string&
grammar_name)
        : DynamicGrammarResultNotification(M_NTFY_INSERTDYNAMICGRAMMARRESULT, callToken,
db_key, result),
          cGrammarName(grammar_name.c_str())
{
}
inline gm::string&
InsertDynamicGrammarResultNotification::GetGrammarName(void)
{
        return cGrammarName;
}
#endif
```

```
/*
 * IPCEndpoint.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:20:54 $
 */
#ifndef IPCENDPOINT_H
#define IPCENDPOINT_H
class __declspec(dllexport) IPCEndpoint
{
        unsigned long cReadMutex;
        unsigned long cWriteMutex;
        bool cUseMutexes;
        bool cNearEndDisconnected;

public:
        IPCEndpoint(void);
        virtual ~IPCEndpoint(void);
        void SetUseMutexes(bool flag);
        inline bool UsingMutexes(void);
        // Server-related methods
        //
        virtual bool Listen(void) = 0;
        virtual IPCEndpoint *ClientConnected(void) = 0;
        virtual IPCEndpoint *WaitForClientConnect(unsigned short NumUserObjects = 0,
                unsigned long *UserObjects = 0) = 0;
        // Client-related methods
        //
        virtual bool Connect(void);
        virtual bool Disconnect(void);
        virtual bool IsConnected(void) = 0;
        // General methods
        //
        bool Write(unsigned char byte,
                unsigned short NumUserObjects = 0, unsigned long *UserObjects = 0);
        bool Read(unsigned char& byte,
                unsigned short NumUserObjects = 0, unsigned long *UserObjects = 0);
        bool Write(unsigned short us,
                unsigned short NumUserObjects = 0, unsigned long *UserObjects = 0);
        bool Read(unsigned short& us,
                unsigned short NumUserObjects = 0, unsigned long *UserObjects = 0);
        bool Write(unsigned long ul,
                unsigned short NumUserObjects = 0, unsigned long *UserObjects = 0);
        bool Read(unsigned long& ul,
                unsigned short NumUserObjects = 0, unsigned long *UserObjects = 0);

        virtual bool Write(unsigned char *buf, unsigned long len,
                unsigned long *BytesWritten,
                unsigned short NumUserObjects = 0, unsigned long *UserObjects = 0) = 0;
        virtual bool Read(unsigned char *buf, unsigned long len,
```

```
                unsigned long *BytesRead,
                unsigned short NumUserObjects = 0, unsigned long *UserObjects = 0) = 0;
        // Synchronization methods
        //
        void BeginAtomicRead(void);
        void EndAtomicRead(void);
        void BeginAtomicWrite(void);
        void EndAtomicWrite(void);
        inline bool NearEndDisconnected(void);
};
inline bool
IPCEndpoint::UsingMutexes(void)
{
        return cUseMutexes;
}
inline bool
IPCEndpoint::NearEndDisconnected(void)
{
        return cNearEndDisconnected;
}
#endif
```

```
/*
 * IPCPipe.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:20:54 $
 */
#ifndef IPCPIPE_H
#define IPCPIPE_H
#include "IPCEndpoint.h"
#include "VPPUtils/GMString"
class __declspec(dllexport) IPCPipe : public IPCEndpoint
{
public:
        gm::string cName;
        HANDLE cPipe;
        OVERLAPPED cReadOverlapped, cWriteOverlapped, cConnectOverlapped;
        bool cError;
        void SetupOverlappedIO(void);
        void Initialize(void);
        IPCPipe(const IPCPipe& other);

public:
        IPCPipe(void);
        IPCPipe(const gm::string& machinename, const gm::string& pipename);
        virtual ~IPCPipe(void);
        void SetMachineAndPipe(const gm::string& machinename, const gm::string& pipename);

        virtual bool Listen(void);
        virtual IPCEndpoint *ClientConnected(void);
        virtual IPCEndpoint *WaitForClientConnect(unsigned short NumUserObjects = 0,
                unsigned long *UserObjects = 0);
        virtual bool Connect(void);
        virtual bool Disconnect(void);
        virtual bool IsConnected(void);
        virtual bool Write(unsigned char *buf, unsigned long len,
                unsigned long *BytesWritten,
                unsigned short NumUserObjects = 0, unsigned long *UserObjects = 0);
        virtual bool Read(unsigned char *buf, unsigned long len,
                unsigned long *BytesRead,
                unsigned short NumUserObjects = 0, unsigned long *UserObjects = 0);
};
#endif
```

```
/*
 * IPCSocket.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.7 $
 * $Date: 2000/12/07 03:20:55 $
 */
#ifndef IPCSocket_H
#define IPCSocket_H
#include <winsock.h>
#include "IPCEndpoint.h"
#include "VPPUtils/GMString"
class __declspec(dllexport) IPCSocket : public IPCEndpoint
{
public:
        static bool cInitialized;
        gm::string cMachine;
        unsigned short cPort;
        SOCKET cSocket;
        HANDLE cConnectedEvent;
        HANDLE cConnectedSocketLatchedEvent;
        HANDLE cTerminateThreadEvent;
        unsigned long cConnectThread;
        SOCKET cConnectedSocket;
        static unsigned __stdcall _ConnectThreadFunc(void *);
        unsigned ConnectThreadFunc(void);
        IPCSocket(const IPCSocket& other, SOCKET s);
        void Initialize(void);
        bool _Disconnect(void);
        void SetSocketOptions(void);
        bool CloseSocket(void);

public:
        IPCSocket(void);
        IPCSocket(const gm::string& machine, unsigned short port);
        virtual ~IPCSocket(void);
        inline void SetMachine(const gm::string& machine);
        inline void SetPort(unsigned short port);

        virtual bool Listen(void);
        virtual IPCEndpoint *ClientConnected(void);
        virtual IPCEndpoint *WaitForClientConnect(unsigned short NumUserObjects = 0,
                unsigned long *UserObjects = 0);
        virtual bool Connect(void);
        virtual bool Disconnect(void);
        virtual bool IsConnected(void);
        virtual bool Write(unsigned char *buf, unsigned long len,
                unsigned long *BytesWritten,
                unsigned short NumUserObjects = 0, unsigned long *UserObjects = 0);
        virtual bool Read(unsigned char *buf, unsigned long len,
                unsigned long *BytesRead,
```

```
                unsigned short NumUserObjects = 0, unsigned long *UserObjects = 0);
};
inline void
IPCSocket::SetMachine(const gm::string& machine)
{
        cMachine = machine.c_str();
}
inline void
IPCSocket::SetPort(unsigned short port)
{
        cPort = port;
}
#endif
```

```
/*
 * Log.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.12 $
 * $Date: 2000/12/07 03:22:24 $
 */
#ifndef _LOG_H
#define _LOG_H
#include "Hashtable.h"
#include "GMString"
class LogFactory;
class __declspec(dllexport) VPPLog {
public:
   static VPPLog& getLogFor(const char* name);
   static LogFactory* setLogFactory(LogFactory* factory);
         VPPLog(void) : enableMask(LOGMASK_ALL) { };
   virtual void va_error(const char *fmt, const va_list vl) = 0;
   virtual void va_warning(const char *fmt, const va_list vl) = 0;
   virtual void va_info(const char *fmt, const va_list vl) = 0;
   virtual void va_diag(const char *fmt, const va_list vl) = 0;
   virtual void va_debug(const char *fmt, const va_list vl) = 0;
   virtual void error(const char *fmt, ...) = 0;
   virtual void warning(const char *fmt, ...) = 0;
   virtual void info(const char *fmt, ...) = 0;
   virtual void diag(const char *fmt, ...) = 0;
   virtual void debug(const char *fmt, ...) = 0;
protected:
         enum {
         LOGMASK_ERROR = 1,
         LOGMASK_WARNING = 2,
         LOGMASK_INFO = 4,
         LOGMASK_DIAG = 8,
         LOGMASK_DEBUG = 16,
         LOGMASK_ALL  = (LOGMASK_ERROR | LOGMASK_WARNING | LOGMASK_INFO |
LOGMASK_DIAG | LOGMASK_DEBUG),
         };
   unsigned int enableMask;
private:
   static LogFactory* factory;
   static Hashtable<char*, VPPLog*>* log_table;
};
class __declspec(dllexport) LogFactory {
public:
   virtual VPPLog* createLog(const char* name) = 0;
};
template<class T>
class __declspec(dllexport) LogT {
public:
   LogT(char* n) {
      name = n;
```

```
    }
    static void error(const char* fmt, ...) {
      check();
      va_list vl;
      va_start(vl, fmt);
      if(log) log->va_error(fmt, vl);
    }
    static void warning(const char* fmt, ...) {
      check();
      va_list vl;
      va_start(vl, fmt);
      if(log) log->va_warning(fmt, vl);
    }
    static void info(const char* fmt, ...) {
      check();
      va_list vl;
      va_start(vl, fmt);
      if(log) log->va_info(fmt, vl);
    }
    static void diag(const char* fmt, ...) {
      check();
      va_list vl;
      va_start(vl, fmt);
      if(log) log->va_diag(fmt, vl);
    }
    static void debug(const char* fmt, ...) {
      check();
      va_list vl;
      va_start(vl, fmt);
      if(log) log->va_debug(fmt, vl);
    }
  private:
    static char* name;
    static VPPLog* log;
    static void check() {
      // no need to synchronize
      if (log == NULL && name != NULL) {
        log = & (VPPLog::getLogFor(name));
      }
    }
};
template<class T>
VPPLog* LogT<T>::log = NULL;
template<class T>
char* LogT<T>::name = NULL;
#endif
```

```
/*
 * LogEnterStateMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:19 $
 */
#ifndef LOGENTERSTATEMSG_H
#define LOGENTERSTATEMSG_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) LogEnterStateMessage : public OneCallTokenMessageBase
{
        gm::string cStateName;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);

public:
        DECLARE_CMSMESSAGE_SUBCLASS(LogEnterStateMessage, OneCallTokenMessageBase);
        inline LogEnterStateMessage(CallToken& ct, const gm::string& StateName);
        inline const gm::string& GetStateName(void);
};
inline
LogEnterStateMessage::LogEnterStateMessage(CallToken& ct, const gm::string& StateName)
        : OneCallTokenMessageBase(M_CMD_LOGENTERSTATE, ct)
{
        cStateName = StateName.c_str();
}
inline const gm::string&
LogEnterStateMessage::GetStateName(void)
{
        return cStateName;
}
#endif
```

```
/*
 * LogGroups.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.6 $
 * $Date: 2000/12/07 03:22:24 $
 */
#ifndef  _LOG_GROUPS_LOG_H
#define  _LOG_GROUPS_LOG_H
#include "VPPUtils/Log.h"
//
//  LogT classes
//
class UtilsLog : public LogT<UtilsLog> {
   UtilsLog() : LogT<UtilsLog>("VPPUtils") {};
   static UtilsLog log; // for initialization
};
class AutoClientLog : public LogT<AutoClientLog> {
   AutoClientLog() : LogT<AutoClientLog>("AutoClient") {};
   static AutoClientLog log; // for initialization
};
#if 0
// Template template
class ????Log : public LogT<????Log> {
   ????Log() : LogT<????Log>("????") {};
   static ????Log log; // for initialization
};
#endif
#endif
```

```
/*
 * LogItemMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:19 $
 */
#ifndef LOGITEMMSG_H
#define LOGITEMMSG_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) LogItemMessage : public OneCallTokenMessageBase
{
        gm::string cItemName;
        gm::string cItemValue;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);

public:
        DECLARE_CMSMESSAGE_SUBCLASS(LogItemMessage, OneCallTokenMessageBase);
        inline LogItemMessage(CallToken& ct, const gm::string& ItemName,
                const gm::string& ItemValue);
        inline const gm::string& GetItemName(void);
        inline const gm::string& GetItemValue(void);
};
inline
LogItemMessage::LogItemMessage(CallToken& ct, const gm::string& ItemName,
        const gm::string& ItemValue) : OneCallTokenMessageBase(M_CMD_LOGITEM, ct)
{
        cItemName = ItemName.c_str();
        cItemValue = ItemValue.c_str();
}
inline const gm::string&
LogItemMessage::GetItemName(void)
{
        return cItemName;
}
inline const gm::string&
LogItemMessage::GetItemValue(void)
{
        return cItemValue;
}
#endif
```

```
/*
 * LogLeaveStateMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:20 $
 */
#ifndef LOGLEAVESTATEMSG_H
#define LOGLEAVESTATEMSG_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
#include "VPPUtils/GMString"
class __declspec(dllexport) LogLeaveStateMessage : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(LogLeaveStateMessage, OneCallTokenMessageBase);
        inline LogLeaveStateMessage(CallToken& token);
};
inline
LogLeaveStateMessage::LogLeaveStateMessage(CallToken& token)
        : OneCallTokenMessageBase(M_CMD_LOGLEAVESTATE, token)
{
}
#endif
```

```
/*
 * MediaChannel.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.44 $
 * $Date: 2000/12/07 03:20:07 $
 */
#ifndef MEDIACHANNEL_H
#define MEDIACHANNEL_H
//#define USE_WFSTREAMER
#include "SCChannel.h"
#include "RecordingMgr.h"
#include "PromptSegmentMgr.h"
#include "SampleTransmitter.h"
#include "SampleListener.h"
#include "VPPUtils/GMLangID.h"
#include "VPPUtils/Synchronize.h"
#include "wfstreamer.h"
class MediaServiceProvider;
class Session;
class SpeechChannel;
class TTSChannel;
class WFStreamer;
class PlayMessage;
class RecordMessage;
class RemoveRecordingMessage;
class HandoffRecordingMessage;
class TTSMessage;
class PlayDTMFMessage;
class CreateCompositePromptMessage;
class AppendCompositePromptPromptMessage;
class AppendCompositePromptSilenceMessage;
class AppendCompositePromptDTMFMessage;
class AppendCompositePromptTTSMessage;
class EndCompositePromptMessage;
class PlayCompositePromptMessage;
class OneCallTokenMessageBase;
class __declspec(dllexport) MediaChannel
        : public SCChannel, public SampleTransmitter,
          public SampleListener
{
public:
        typedef enum
        {
                S_Idle,
                S_Playing,
                S_Recording,
                S_StopPlayRequested,
                S_StopRecordRequested,
                S_PlayingTTS
        } ChannelState;
```

```
#ifdef ENABLE_PERF_MEASURE
protected: // timing
        unsigned long cStartRecordingTime;
        unsigned long cStartSendingRecSamplesTime;
        unsigned long cDoStopTime;
#endif
private:
        typedef enum
        {
                ePSS_Failure,
                ePSS_Playing,
                ePSS_AllSegmentsDone
        } PromptSegmentStatus;
        CriticalSection    cStateLock;
        SpeechChannel   *pSpChanTBNotified;     // speech channel to be notified *only* when playing is done

        inline MediaServiceProvider *GetMSP(void);
        MediaChannel::PromptSegmentStatus PlaySegmentRun(bool First, CallToken& ct);
        void CleanupRecordings(void);
        unsigned long StopAndNotify(void);
protected:
        bool cPlayingComposite;
        bool cStoppingCompositePlay;
        bool cCompositePlayStopped;
        ChannelState cState;
        CMSMessage::SerialNumberT cMsgSerialNumber; // used currently only for NotifyFailure
        gm::string cPlayingFilename;
        // the following are used for TTS streaming play
        FILE            *cpWriteStreamingPlayFile;      // the 'write' file pointer to the buffer file in the case of
'streaming' play
        HANDLE                  cEndOfStreamPlayDataEvent;                      // Signaled when
NotifyEndOfData is called
        HANDLE                  cStreamingWriteMutex;
        RecordingManager cRecordingMgr;
        PromptSegmentManager cPromptSegmentMgr;
        SampleListener *cpListener;

        //
        // Pure-virtual methods that must be implemented by the subclass.  All
        // return one of the MSPF_xxx codes from vppmsg.h
        //
        typedef enum
        {
                MCPlay_File,
                MCPlay_Stream,
                MCPlay_TTS,
        } MChannelPlayType;
        virtual unsigned long Play(const gm::string& filename, bool ttsSource, bool useCallback) = 0;
        virtual unsigned long PlaySilence(unsigned long durationmsecs) = 0;
   virtual unsigned long PlayDTMF(const gm::string& dtmfString) = 0;
        void StopTTS(void);     // This should probably be protected.
        //
        // Notification methods that derived classes call to notify the
        // base class of asynchronous events
        //
        void NotifyPlayDone(void);
```

```
        void NotifyStopped(void);
        void NotifyDTMF(char dtmf);
        inline SampleListener *GetSampleListener(void);
public:
        inline MediaChannel(ServiceProvider *pServiceProvider, const char *logType, int id, bool
RequiresSettlement = false);
        virtual ~MediaChannel(void);
        inline ChannelState GetState(void);
        inline void SetState(MediaChannel::ChannelState state);
        virtual unsigned long Stop(void) = 0;
        virtual unsigned long Record(const gm::string& filename, bool bStreaming, int bos_timeout, int tms_timeout,
int eos_timeout) = 0; // all timeouts in 10msecs
        unsigned long DoPlay(CallToken& ct,
                const gm::string& filename, bool IsTTS, bool useCallback);
        unsigned long DoPlayTTS(CallToken& ct, const gm::string& ttsText);
        unsigned long DoPlaySilence(CallToken& ct, unsigned long durationmsecs);
        unsigned long DoPlayDTMF(CallToken& ct, const gm::string& DTMFString);
        unsigned long DoStop(void);
        unsigned long DoAppendCompositePromptTTS(CallToken& ct,
                const gm::string& product, GMLANGID lang, const gm::string& text);
        virtual bool GetRecordingFilename(RECORDINGID recordingID,
                gm::string& filename);
  inline RecordingManager& GetRecordingManager(void);

        bool StopCompositePlay();
        // sample transmitter methods
        bool StartTransmission(void);
        bool StopTransmission(void);
        inline bool SetSampleListener(SampleListener *pListener);
        // sample listener methods
        void NotifySample(const unsigned char *pSample, unsigned int numBytes);
        void NotifyEndOfData(int reason);
        // Request for Notification of play done and/or dtmf
        bool RequestNotification(SpeechChannel *pSpeechChannel);
        void NotifyFailure(void);
        virtual void NotifyPropertySet(Property& prop);
        virtual void NotifyBoundToSession(SessionNode *pSessionNode);
        inline void GetRecordingFilename(gm::string& filename);
#ifdef ENABLE_PERF_MEASURE
        inline void SetStartRecordingTime(unsigned long time);
        inline unsigned long GetStartRecordingTime(void);
        inline void SetDoStopTime(unsigned long time);
        inline unsigned long GetDoStopTime(void);
#endif
// new media api
        virtual bool HandlePlay(SessionNode *pSession, PlayMessage *pMsg, MediaChannel *pRecordChannel, void
*userdata);
        virtual bool HandleStop(SessionNode *pSession, OneCallTokenMessageBase *pMsg, void *userdata);
        virtual bool HandleRecord(SessionNode *pSession, RecordMessage *pMsg, void *userdata);
        virtual bool HandleRemoveRecording(SessionNode *pSession, RemoveRecordingMessage *pMsg, void
*userdata);
        virtual bool HandleReleaseRecording(SessionNode *pSession, CallToken& callToken, RECORDINGID
recordingID);
        virtual bool HandleTTS(SessionNode *pSession, TTSMessage *pMsg, void *userdata);
        virtual bool HandleCreateCompositePrompt(SessionNode *pSession, CreateCompositePromptMessage
*pMsg, void *userdata);
```

```
        virtual bool HandleAppendCompositePromptPrompt(SessionNode *pSession,
AppendCompositePromptPromptMessage *pMsg, void *userdata);
        virtual bool HandleAppendCompositePromptSilence(SessionNode *pSession,
AppendCompositePromptSilenceMessage *pMsg, void *userdata);
        virtual bool HandleAppendCompositePromptDTMF(SessionNode *pSession,
AppendCompositePromptDTMFMessage *pMsg, void *userdata);
        virtual bool HandleAppendCompositePromptTTS(SessionNode *pSession,
AppendCompositePromptTTSMessage *pMsg, void *userdata);
        virtual bool HandleEndCompositePrompt(SessionNode *pSession, EndCompositePromptMessage *pMsg,
void *userdata);
        virtual bool HandlePlayCompositePrompt(SessionNode *pSession, PlayCompositePromptMessage *pMsg,
void *userdata);
        virtual bool NotifyCallAboutToBeDropped(SessionNode *pSession, CallToken& callToken);
// end new media api
};
inline
MediaChannel::MediaChannel(ServiceProvider *pServiceProvider, const char *logType, int id, bool
RequiresSettlement)
        : SCChannel(pServiceProvider, Channel::CT_Media, logType, id, RequiresSettlement),
          cpWriteStreamingPlayFile(NULL), pSpChanTBNotified(NULL),
          cStoppingCompositePlay(false), cCompositePlayStopped(true)
{
        cState = S_Idle;
        cPlayingComposite = false;
        pSpChanTBNotified = 0;
        cEndOfStreamPlayDataEvent = ::CreateEvent(NULL, true, false, NULL);  // must be manually reset!
        cStreamingWriteMutex = CreateMutex(NULL, false, NULL);
}
inline MediaChannel::ChannelState
MediaChannel::GetState(void)
{
  return cState;
}
inline void
MediaChannel::SetState(MediaChannel::ChannelState state)
{
        cState = state;
}
inline MediaServiceProvider *
MediaChannel::GetMSP(void)
{
        return (MediaServiceProvider *)cServiceProvider;
}
inline RecordingManager&
MediaChannel::GetRecordingManager(void)
{
        return cRecordingMgr;
}
inline
bool
MediaChannel::SetSampleListener(SampleListener *pListener)
{
        cpListener = pListener;
        return true;
}
inline SampleListener *
```

```
MediaChannel::GetSampleListener(void)
{
        return cpListener;
}
inline void
MediaChannel::GetRecordingFilename(gm::string& filename)
{
        filename = "";
}
#ifdef ENABLE_PERF_MEASURE
inline void
MediaChannel::SetStartRecordingTime(unsigned long time)
{
        cStartRecordingTime = time;
}
inline unsigned long
MediaChannel::GetStartRecordingTime(void)
{
        return cStartRecordingTime;
}
inline void
MediaChannel::SetDoStopTime(unsigned long time)
{
        cDoStopTime = time;
}
inline unsigned long
MediaChannel::GetDoStopTime(void)
{
        return cDoStopTime;
}
#endif
#endif
```

```
/*
 * MediaServiceProvider.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.24 $
 * $Date: 2000/12/07 03:20:07 $
 */
#ifndef MEDIASERVICEPROVIDER_H
#define MEDIASERVICEPROVIDER_H
//#define USE_WFSTREAMER
#include "ServiceProvider.h"
#include "VPPMsg/VPPDefs.h"
#include "wfstreamer.h"
#include "VPPUtils/CriticalSection.h"
class MediaChannel;
class MTSServer;
class PlayMessage;
class StopPlayMessage;
class RecordMessage;
class StopRecordMessage;
class RemoveRecordingMessage;
class HandoffRecordingMessage;
class TTSMessage;
class CreateCompositePromptMessage;
class AppendCompositePromptPromptMessage;
class AppendCompositePromptSilenceMessage;
class AppendCompositePromptTTSMessage;
class EndCompositePromptMessage;
class PlayCompositePromptMessage;
class HandOffRecordingMessage;
class RecordingHandler;
class __declspec(dllexport) MediaServiceProvider : public ServiceProvider
{
        struct RecordingHandlerNode
        {
#if _MSC_VER < 1200
                HANDLE cHdl;
#else
                HINSTANCE__ *cHdl;
#endif
                RecordingHandler *cRecordingHandler;
        };
        template class __declspec(dllexport) std::list<RecordingHandlerNode *>;
        std::list<RecordingHandlerNode *> cRecordingHandlers;
public:
        MediaServiceProvider(MTSServer *pServer, const char *logType);
        virtual ~MediaServiceProvider(void);
        virtual bool Teardown(void);
        bool HandOffRecording(const gm::string& ServiceName,
                const gm::string& filename, void *pData, unsigned long DataLength);
        // Handlers - methods that handle requests and notifications from
```

```
        // clients
        //
};
typedef MediaServiceProvider *(*CREATEMSPFUNC)(MTSServer *);
extern "C" __declspec(dllexport)
MediaServiceProvider *CreateMediaServiceProvider(MTSServer *);
#endif
```

Document ID: 1

```
/*
 * Monitor.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.8 $
 * $Date: 2000/12/07 03:22:24 $
 */
#ifndef _MONITOR_H
#define _MONITOR_H
#include <windows.h>
class __declspec(dllexport) Monitor {
public:
   Monitor(void);
   virtual ~Monitor(void);
private:
   friend class Synchronized;
   void enter(void);
   void wait(long duration=0);
   void notify();
   void notifyAll();
   void leave(void);
   int  yield();
   void join(int nest);
   unsigned long owner;
   HANDLE mutex;
   unsigned long thisThread();
   HANDLE getFromPool();
   void returnToPool(HANDLE event);
   void removeFromWaiting(HANDLE event);
   void queue(HANDLE event);
   HANDLE dequeue();
   HANDLE pool[10];
   HANDLE waiting[10];
   int top, tail;
   int poolsize;
   int nest;
   bool dead;
};
#endif
```

```
/*
 * MTS.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.10 $
 * $Date: 2000/12/07 03:23:31 $
 */
#ifndef MTS_H
#define MTS_H
#include "MTSLib/MTSServer.h"
#include "MTSLib/MTSServerImpl.h"
class MTS
{
   MTSServer cMTSServer;
   MTSServerImpl corbaServer;
   // bool RegisterServers(bool up);
   CORBA::ORB_var cORB;
   PortableServer::POA_var rootPOA;
   gm::string group;
   boolean error;
public:
        static HANDLE cExitEvent;
public:
   MTS(const gm::string& name, const gm::string& clientgroup, const gm::string& PromptRoot);
   ~MTS(void);
   bool Initialize(int argc, char** argv);
   bool Go(void);
   void Loop();
   inline MTSServer *GetMTSServer(void);
};
inline MTSServer *
MTS::GetMTSServer(void)
{
        return &cMTSServer;
}
#endif
```

```
/*
 * MTSDefs.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:20:06 $
 */
#ifndef MTSDEFS_H
#define MTSDEFS_H
/*
 * Pipe name for IPC
 */
#define MTS_PIPENAME "gmMTS"
/*
 * Default port for socket IPC
 */
#define MTS_DEFAULTPORT 3579
#endif
```

EXHIBIT 2002

```
/*
 * MTSDownNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:20 $
 */
```

Docket No. 3000.1

```
/*
 * MTSMonitorImpl.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction controlled under your contract with General Magic
 *
 * $Revision: 1.1 $
 * $Date: 2000/10/23 01:47:52 $
 *
 * $Id: MTSMonitorImpl.h,v 1.1 2000/10/23 01:47:52 Dmitri Khokhlov Exp $
 */
#ifndef _MTSMONITORIMPL_H
#define _MTSMONITORIMPL_H
#include "MTSLib/invui_s.h"
#include "MTSLib/mtsmon_s.h"
#include "VPPUtils/Monitor.h"
#include "VPPUtils/Thread.h"
class MTSServer;
class TelephonyChannel;
class __declspec(dllexport) MTSMonitorImpl : public POA_mtsmon::MTSMonitor, public Monitor,
                             public Thread
{
   MTSServer*              _pMTSServer;
   mtsmon::MTSMonitorAgent_ptr _cAgent;
   invui::MTSServer_ptr       _cMTSServer;
   DWORD                   _threadId;
         HANDLE                                                    _hMtsExitEvent;
static DWORD m_start_time;
public:
   MTSMonitorImpl( MTSServer* );
   virtual ~MTSMonitorImpl();
         // access _cMTSServer
   virtual invui::MTSServer_ptr mts () { return _cMTSServer; };
   virtual void mts ( invui::MTSServer_ptr mts ) { _cMTSServer = mts; };
         // access _cMTSServer
         MTSServer* server () { return _pMTSServer; };
         // activate this object as CORBA Servant
         int activate ( PortableServer::POA_var rootPOA, HANDLE hExitEvent );
   TelephonyChannel* getTelephonyChannel( long ch );
   virtual void ping ();
   virtual void run (); // message loop
   virtual BOOL postMessage ( UINT Msg, WPARAM wParam, LPARAM lParam );
         // CORBA methods implementation
   virtual CORBA::Long registerAgent ( mtsmon::MTSMonitorAgent_ptr ma );
   virtual CORBA::Long channelControl ( CORBA::Long ch, CORBA::Long cmd );
   virtual CORBA::Long serviceControl ( CORBA::Long cmd );
   virtual CORBA::Long getServerData ( mtsmon::MTSMonitorServerData& dat);
   virtual CORBA::Long getChannelData (CORBA::Long _ch, mtsmon::MTSMonitorChannelData& _cd);
};
#endif
```

```
/*
 * MTSServer.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.24 $
 * $Date: 2000/12/07 03:20:07 $
 */
#ifndef MTSSERVER_H
#define MTSSERVER_H
#include "VPPMsg/VPPDefs.h"
#include "VPPUtils/Monitor.h"
#include "SessionMgr.h"
#include "VPPMsg/PropertyList.h"
#include "VPPUtils/Log.h"
#include "Regex/RegularExpression.h"
#include <list>
class MTS;
class TelephonyServiceProvider;
class MediaServiceProvider;
class TTSServiceProvider;
class SpeechServiceProvider;
class TelephonyChannel;
class CallToken;
class RecordingHandler;
class MTSServerImpl;
class MTSMonitorImpl;
class __declspec(dllexport) MTSServer : public Monitor
{
        HINSTANCE chTSPDLL;
        TelephonyServiceProvider *cTSP;
        HINSTANCE chMSPDLL;
        MediaServiceProvider *cMSP;
        HINSTANCE chTTSSPDLL;
        TTSServiceProvider *cTTSSP;
        HINSTANCE chSSPDLL;
        SpeechServiceProvider *cSSP;
        SessionMgr cSessionMgr;
        PropertyList cSystemProperties;
private:
  MTSMonitorImpl* cMonitor;

#ifdef OSHIMA
friend SessionNode;
  TelephonyControlImpl* createTelephonyControl(SessionNode* s);
#endif
public:
        MTSServer(const gm::string& PromptRoot, MTSServerImpl* impl);
        virtual ~MTSServer(void);
        inline TelephonyServiceProvider *GetTelephonyServiceProvider(void);
        inline MediaServiceProvider *GetMediaServiceProvider(void);
        inline TTSServiceProvider *GetTTSServiceProvider(void);
```

```
        inline SpeechServiceProvider *GetSpeechServiceProvider(void);
        inline SessionMgr *GetSessionMgr(void);
        inline PropertyList& GetSystemProperties(void);
public:
   bool Initialize();
   bool PostMonitorMessage( UINT Msg, WPARAM wParam, LPARAM lParam );
   MTSMonitorImpl* GetMTSMonitor();
};
inline TelephonyServiceProvider *
MTSServer::GetTelephonyServiceProvider(void)
{
        return cTSP;
}
inline MediaServiceProvider *
MTSServer::GetMediaServiceProvider(void)
{
        return cMSP;
}
inline TTSServiceProvider *
MTSServer::GetTTSServiceProvider(void)
{
        return cTTSSP;
}
inline SpeechServiceProvider *
MTSServer::GetSpeechServiceProvider(void)
{
        return cSSP;
}
inline SessionMgr *
MTSServer::GetSessionMgr(void)
{
        return &cSessionMgr;
}
inline PropertyList&
MTSServer::GetSystemProperties(void)
{
        return cSystemProperties;
}
class MTSLibLog : public LogT<MTSLibLog> {
   MTSLibLog() : LogT<MTSLibLog>("MTS_LIB ") {};
   static MTSLibLog log; // for initialization
};
#endif
```

```
/*
 * MTSServerImpl.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:20:07 $
 */
#ifndef _MTSSERVERIMPL_H
#define _MTSSERVERIMPL_H
#include <VPPUtils/GMSTRING>
#include "invui_s.h"
class TelephonyServiceProvider;
class __declspec(dllexport) MTSServerImpl : public POA_invui::MTSServer
{
  gm::string cClientGroupName;
  TelephonyServiceProvider* cTSP;
  CORBA::ORB_ptr cORB;
  /*********************************************************/
  //
  // CORBA METHODS
  //
  /*********************************************************/
  virtual invui::Call* placeCall(invui::VUIClient_ptr vc,
                    const char* nubmer,
                    const char* ani,
                                                    const invui::UUIType& uui,
                    invui::TIME timeout);
  virtual void ping();
public:
  MTSServerImpl(const gm::string &clientname);
  virtual ~MTSServerImpl(void) {}
  // inline CORBA::ORB_ptr GetORB(void);
  inline void MTSServerImpl::SetTSP(TelephonyServiceProvider* tsp) {
    cTSP = tsp;
  };
  inline CORBA::ORB_ptr GetORB(void) {
    return cORB;
  }
  void SetORB(CORBA::ORB_ptr orb) {
    cORB = orb;
  }
  invui::VUIClient_ptr GetVUIClient();
};
#endif
```

```
/*
 * NLInterpretation.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:20 $
 */
#ifndef NLInterpretation_H
#define NLInterpretation_H
#include "Slot.h"
#include <list>
#pragma warning(disable:4786)
#pragma warning(disable:4660)
class __declspec(dllexport) NLInterpretation
{
        template class __declspec(dllexport) std::list<Slot *>;
        std::list<Slot *> cSlots;
        gm::string grammar_key;
public:
        NLInterpretation(void);
        ~NLInterpretation(void);
        unsigned short GetNumSlots(void);
        Slot *GetSlot(int index);
        gm::string& GetGrammarKey();
        void SetGrammarKey(gm::string& key);
        void AddSlot(Slot *pSlot);
        void Serialize(VPPMessage *);
        void Deserialize(VPPMessage *);
};
#endif
```

```
/*
 * NotifySampleEvent.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:25:09 $
 */
#ifndef _NOTIFY_SAMPLE_EVENT_H
#define _NOTIFY_SAMPLE_EVENT_H
#include "MTSLib/SampleListener.h"
class SampleListener;
class __declspec(dllexport) NotifySampleEvent {
  unsigned char* buf;
  unsigned int size;
  SampleListener* listener;
public:
  NotifySampleEvent(SampleListener* l,
                    unsigned char* src_buf,
                    unsigned int s) : listener(l)
  {
    size = s;
    buf = new unsigned char[size];
    memcpy(buf, src_buf, size * sizeof(unsigned char));
  }
  ~NotifySampleEvent() {
    delete [] buf;
  }
  void notifySample() {
    listener->NotifySample((const unsigned char*)buf,
                           (const unsigned int)size);
  }
};
#endif
```

```
/*
 * NuanceSChannel.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.22 $
 * $Date: 2000/12/07 03:24:28 $
 */
#ifndef NUANCESCHANNEL_H
#define NUANCESCHANNEL_H
#include <MTSLib/SpeechChannel.h>
#include "nuance-config.h"
#include <VPPMsg/RecognitionDoneNtfy.h>
#include <MTSLib/DynamicGrammar.h>
class VRSAPIClient;
class RecognitionResults;
class ConfigFile;
class MediaChannel;
class Dispatcher;
class SessionNode;
class __declspec(dllexport) NuanceSChannel : public SpeechChannel
{
        friend class VRSAPIClient;
public:
        typedef enum
        {
                S_Nonfunctional,
                S_Unlicensed,
                S_Idle,
                S_Recognizing,
                S_WaitingForResults
        } State;
private:
  enum {
    DG_NONE,
    DG_QUERY,
    DG_NOTFOUND,
    DG_CREATE,
    DG_INSERT,
    DG_SUCCESS,
    DG_ERROR,
  } cDynamicGrammarState;
  int cNumberOfInsertedGrammars;
  Hashtable<char*, GrammarSet*>* cGrammarSets;
  unsigned long InsertGrammars(const GrammarHandle& handle);
  static char* cDynamicGrammarNames[26];
  static char* cDynamicGrammarRefs[26];
  NuanceStatus cNuanceStatus;
  gm::string cTopLevelGrammar;
  unsigned long NuanceToSSPF(NuanceStatus status);
        State cState;
        VRSAPIClient *cVRSAPI;
```

```
protected:
        //
        // From SpeechChannel
        //
#if 0
        virtual unsigned long StartRecognition(const gm::string& grammar);
        virtual unsigned long RecognizeString(const gm::string& grammar,
                const gm::string& str);
#else
        virtual unsigned long StartRecognition(const GrammarHandle& grammar);
        virtual unsigned long RecognizeString(const GrammarHandle& grammar,
                const gm::string& str);
#endif
    virtual unsigned long GetGrammarHandle(DynamicGrammar& gram,
                            GrammarHandle& handle);
    unsigned long GetGrammarSetHandle(const GrammarSet& gram,
                          GrammarHandle& handle);
    unsigned long GetGrammarHandle(const gm::string& key,
                        const gm::string& gsl,
                        GrammarHandle& handle);
        virtual unsigned long PushSamples(const unsigned char *pSamples, unsigned int numbytes);
        virtual unsigned long StopRecognition(void);
        virtual unsigned long AbortRecognition(void);
        virtual bool StartLogging(void);
        virtual bool LogUtteranceFilename(void);
        virtual bool StopLogging(void);
#if 0
        virtual unsigned long CreateDynamicGrammar(const gm::string& db_key, const gm::string& gsl);
        virtual unsigned long DeleteDynamicGrammar(const gm::string& db_key);
        virtual unsigned long AddPhraseToDynamicGrammar(const gm::string& db_key,
                const gm::string& phrase_id, const gm::string& phrase_text,
                const gm::string& phrase_nl, const unsigned long probability);
        virtual unsigned long RemovePhraseFromDynamicGrammar(const gm::string& db_key,
                const gm::string& phrase_id);
        virtual unsigned long QueryDynamicGrammarContents(const gm::string& db_key,
                const gm::string& phrase_id);
        virtual unsigned long InsertDynamicGrammar(const gm::string& db_key,
                const gm::string& grammar_name);
#else
  /* from SpeechChannel.h */
  void NotifyDynamicGrammarCreated(NuanceStatus status,
                      const gm::string& db_key);
  void NotifyDynamicGrammarDeleted(NuanceStatus status,
                      const
                      gm::string& db_key);
  void NotifyPhraseAddedToDynamicGrammar(NuanceStatus status,
                          const gm::string& db_key,
                          const gm::string& phrase_id);
  void NotifyPhraseRemovedFromDynamicGrammar(NuanceStatus status,
                            const gm::string& db_key,
                            const gm::string& phrase_id);
  void NotifyDynamicGrammarContents(NuanceStatus status,
                      const gm::string& db_key,
                      const gm::string& phrase_id,
                      const gm::string& phrase_contents);
  void NotifyDynamicGrammarInserted(NuanceStatus status,
```

```
                        const gm::string& db_key,
                        const gm::string& grammar_name);
#endif
public:
        NuanceSChannel(ServiceProvider *pServiceProvider, const char *logType, int id);
        virtual ~NuanceSChannel(void);
        bool Setup(unsigned int id, NuanceConfig *pConfig);
        void NotifyDoneRecognizing(RecognitionResults *pResults,
                RecognitionDoneNotification::RecognitionStatus status);
        inline void NotifyStartOfSpeech(void);
        inline void NotifyEndOfSpeech(void);
        inline MediaChannel *GetRecordChannel(void);
        virtual void NotifyBoundToSession(SessionNode *pSessionNode);
        virtual void NotifyPropertySet(Property& prop);
        void NotifySettled(void);
    inline SessionNode* GetSession();
};
inline void
NuanceSChannel::NotifyStartOfSpeech(void)
{
        SpeechChannel::NotifyStartOfSpeech();
}
inline void
NuanceSChannel::NotifyEndOfSpeech(void)
{
        SpeechChannel::NotifyEndOfSpeech();
}
inline SessionNode*
NuanceSChannel::GetSession(void)
{
  return cSession;
}
// Log class
class NUAN_SSPLog : public LogT<NUAN_SSPLog> {
  NUAN_SSPLog() : LogT<NUAN_SSPLog>("NUAN_SSP") {};
  static NUAN_SSPLog log; // for initialization
};
#endif
```

```
/*
 * NuanceSSP.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.8 $
 * $Date: 2000/12/07 03:24:28 $
 */
#ifndef NUANCESSP_H
#define NUANCESSP_H
#include "MTSLib/SpeechServiceProvider.h"
#include "VPPUtils/GMString"
class NuanceSChannel;
class __declspec(dllexport) NuanceSSP : public SpeechServiceProvider
{
private:
        static const char const *cLogType;
        static NuanceSSP *cInstance;
        static gm::string cPackagesSpecificationFile;
        static gm::string cParametersFile;
        static bool cLogRecordingFilename;

public:
        NuanceSSP(MTSServer *pServer);
        virtual ~NuanceSSP(void);
        virtual bool Setup(void);
        virtual bool Teardown(void);
        static NuanceSSP *GetInstance(void);
        static void SetInstance(NuanceSSP *ptr);
        inline static bool GetLogRecordingFilename(void);
};
inline bool
NuanceSSP::GetLogRecordingFilename(void)
{
        return cLogRecordingFilename;
}
#endif
```

```
/*
 * OneCallTokenMsgBase.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.6 $
 * $Date: 2000/12/07 03:21:20 $
 */
#ifndef ONECALLTOKENMSGBASE_H
#define ONECALLTOKENMSGBASE_H
#include "VPPMsg.h"
#include "CallToken.h"
class __declspec(dllexport) OneCallTokenMessageBase : public VPPMessage
{
protected:
        CallToken cCallToken;
        virtual void Serialize(void);
        virtual void Deserialize(void);
        inline OneCallTokenMessageBase(unsigned short cmdid);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(OneCallTokenMessageBase, VPPMessage);
        inline OneCallTokenMessageBase(unsigned short cmdid, CallToken& CallToken);
        inline CallToken& GetCallToken(void);
#if 0
        virtual VirtualMachineID& GetMTSVirtualMachineID(void);
#endif
};
inline
OneCallTokenMessageBase::OneCallTokenMessageBase(unsigned short cmdid)
        : VPPMessage(cmdid)
{
}
inline
OneCallTokenMessageBase::OneCallTokenMessageBase(unsigned short cmdid,
        CallToken& CallToken) : VPPMessage(cmdid)
{
        cCallToken = CallToken;
}
inline CallToken&
OneCallTokenMessageBase::GetCallToken(void)
{
        return cCallToken;
}
#endif
```

```
/*
 * OutboundCallRequest.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:20:08 $
 */
#ifndef _OUTBOUND_CALL_REQUEST_H
#define _OUTBOUND_CALL_REQUEST_H
#include "invui_s.h"
#include "VPPMsg/CallProgressNtfy.h"
#include "VPPMsg/RequestToken.h"
#include "VPPUtils/Monitor.h"
#include "VPPUtils/Hashtable.h"
#include "VPPUtils/Queue.h"
#include "invui_c.h"
#include "ConnectionListener.h"
class VUISessionImpl;
class OutboundCallRequest : public ConnectionListener, public Monitor {
   invui::Call_ptr call;
   CallProgressNotification::CallProgressT status;
   RequestToken token;
   bool hasResult;
   VUISessionImpl* session;
   static REQUESTID GetNextRequestID();
   static Queue<OutboundCallRequest*> zombieList;
   static Hashtable<RequestToken*, OutboundCallRequest*>* table;
   bool RemoveFromTable();
   OutboundCallRequest();
   virtual ~OutboundCallRequest();
public:
   static void Destroy(OutboundCallRequest* req);
   static OutboundCallRequest* GetOutboundCallRequest(RequestToken& token);
   static OutboundCallRequest* Create();
   void Disconnected();
   void SetSession(VUISessionImpl* s);
   VUISessionImpl* GetSession();
   RequestToken& GetRequestToken();
   void CallAlerting(invui::Call_ptr c);
   void CallFailure(CallProgressNotification::CallProgressT s);
   invui::Call_ptr GetCall(long timeout = 0);
};
inline RequestToken&
OutboundCallRequest::GetRequestToken()
{
   return token;
}
#endif
```

```
/*
 * PlaceCallMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.10 $
 * $Date: 2000/12/07 03:21:21 $
 */
#ifndef PLACECALLMSG_H
#define PLACECALLMSG_H
#include "VPPDefs.h"
#include "Address.h"
#include "OneCallTokenMsgBase.h"
#include "RequestToken.h"
#include "CDRInfo.h"
class PropertyList;
class __declspec(dllexport) PlaceCallMessage : public OneCallTokenMessageBase
{
        Address cTarget;
        Address cCallingNumber;
        RequestToken cRequest;
        PropertyList *cProperties;
        bool cQueue;
        bool cOwned;
        CDRInfo cCDRInfo;
        inline void init(Address& target, Address& callingNumber,
                RequestToken& request, PropertyList *pProps, CDRInfo& cdrinfo, bool queue);
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);

public:
        DECLARE_CMSMESSAGE_SUBCLASS(PlaceCallMessage, OneCallTokenMessageBase);
        inline PlaceCallMessage(Address& target, Address& callingNumber,
                RequestToken& request, PropertyList *pProps, CDRInfo& cdrinfo, bool queue = false);
        inline PlaceCallMessage(Address& target, Address& callingNumber,
                RequestToken& request, CallToken& callToken, PropertyList *pProps,
                CDRInfo& cdrinfo, bool queue = false);
        PlaceCallMessage(const PlaceCallMessage& other);
        virtual ~PlaceCallMessage(void);
        inline void SetTransferTarget(CallToken& callToken);
        inline Address& GetTarget(void);
        inline Address& GetCallingNumber(void);
        inline RequestToken& GetRequestToken(void);
        inline CallToken& GetTransferTarget(void);
        inline PropertyList *GetProperties(void);
        inline bool GetQueue(void);
        inline CDRInfo& GetCDRInfo(void);
#if 0
        virtual VirtualMachineID& GetMTSVirtualMachineID(void);
#endif
};
```

```
inline void
PlaceCallMessage::init(Address& target, Address& callingNumber,
        RequestToken& request, PropertyList *pProps, CDRInfo& cdrinfo,
        bool queue)
{
        cTarget = target;
        cCallingNumber = callingNumber;
        cRequest = request;
        cProperties = pProps;
        cOwned = false;
        cQueue = queue;
        cCDRInfo = cdrinfo;
}
inline
PlaceCallMessage::PlaceCallMessage(Address& target,
        Address& callingNumber, RequestToken& request, PropertyList *pProps,
        CDRInfo& cdrinfo, bool queue)
        : OneCallTokenMessageBase(M_CMD_PLACECALL)
{
        init(target, callingNumber, request, pProps, cdrinfo, queue);
}
inline void
PlaceCallMessage::SetTransferTarget(CallToken& callToken)
{
        cCallToken = callToken;
}
inline
PlaceCallMessage::PlaceCallMessage(Address& target,
        Address& callingNumber, RequestToken& request, CallToken& TransferTarget,
        PropertyList *pProps, CDRInfo& cdrinfo, bool queue)
        : OneCallTokenMessageBase(M_CMD_PLACECALL, TransferTarget)
{
        init(target, callingNumber, request, pProps, cdrinfo, queue);
        SetTransferTarget(TransferTarget);
}
inline Address&
PlaceCallMessage::GetTarget(void)
{
        return cTarget;
}
inline Address&
PlaceCallMessage::GetCallingNumber(void)
{
        return cCallingNumber;
}
inline RequestToken&
PlaceCallMessage::GetRequestToken(void)
{
        return cRequest;
}
inline CallToken&
PlaceCallMessage::GetTransferTarget(void)
{
        return GetCallToken();
}
inline PropertyList *
```

```
PlaceCallMessage::GetProperties(void)
{
        return cProperties;
}
inline bool
PlaceCallMessage::GetQueue(void)
{
        return cQueue;
}
inline CDRInfo&
PlaceCallMessage::GetCDRInfo(void)
{
        return cCDRInfo;
}
#endif
```

Document ID: 1000000000000000

```
/*
 * PlaceCallOnHoldMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:21 $
 */
#ifndef PLACECALLONHOLDMSG_H
#define PLACECALLONHOLDMSG_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) PlaceCallOnHoldMessage : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(PlaceCallOnHoldMessage,
                OneCallTokenMessageBase);
        inline PlaceCallOnHoldMessage(CallToken& token);
};
inline
PlaceCallOnHoldMessage::PlaceCallOnHoldMessage(CallToken& token)
        : OneCallTokenMessageBase(M_CMD_PLACECALLONHOLD, token)
{
}
#endif
```

```
/*
 * PlayCompositePromptMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:21 $
 */
#ifndef PLAYCOMPOSITEPROMPTMSG_H
#define PLAYCOMPOSITEPROMPTMSG_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) PlayCompositePromptMessage
        : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(PlayCompositePromptMessage,
                OneCallTokenMessageBase);
        inline PlayCompositePromptMessage(CallToken& token);
};
inline
PlayCompositePromptMessage::PlayCompositePromptMessage(
        CallToken& token)
        : OneCallTokenMessageBase(M_CMD_PLAYCOMPOSITEPROMPT, token)
{
}
#endif
```

Document ID: 2000P2000P2000P2000P2000P2000P2000P2000P

```
/*
 * PlayDoneNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:21 $
 */
#ifndef PLAYDONENTFY_H
#define PLAYDONENTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) PlayDoneNotification : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(PlayDoneNotification,
                OneCallTokenMessageBase);
        inline PlayDoneNotification(CallToken& token);
};
inline
PlayDoneNotification::PlayDoneNotification(CallToken& token)
        : OneCallTokenMessageBase(M_NTFY_PLAYDONE, token)
{
}
#endif
```

```
/*
 * PlayingNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:22 $
 */
#ifndef PLAYINGNTFY_H
#define PLAYINGNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) PlayingNotification : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(PlayingNotification,
                OneCallTokenMessageBase);
        inline PlayingNotification(CallToken& token);
};
inline
PlayingNotification::PlayingNotification(CallToken& token)
        : OneCallTokenMessageBase(M_NTFY_PLAYING, token)
{
}
#endif
```

```
/*
 * PlayMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.7 $
 * $Date: 2000/12/07 03:21:22 $
 */
#ifndef PLAYMSG_H
#define PLAYMSG_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
#include "VPPUtils/GMString"
#include "VPPUtils/GMLangID.h"
class __declspec(dllexport) PlayMessage : public OneCallTokenMessageBase
{
        gm::string cProductName;
        gm::string cPromptName;
        RECORDINGID cRecordingID;
        GMLANGID cLangID;
        gm::string cTTSAlternate;
        bool cUseSearchPath;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(PlayMessage, OneCallTokenMessageBase);
        inline PlayMessage(CallToken& token, const gm::string& promptname,
                const gm::string& productname, GMLANGID lang,
                const gm::string& tts_alternate);
        inline PlayMessage(CallToken& token, RECORDINGID recordingID);
        inline PlayMessage(CallToken& token, const gm::string& promptname,
                const gm::string& tts_alternate, bool bUseSearchPath=false);
        //inline PlayMessage(CallToken& token, const gm::string& fullpath,
        //      const gm::string& tts_alternate);
        inline const gm::string& GetProductName(void);
        inline const gm::string& GetPromptName(void);
        inline GMLANGID GetLangID(void);
        inline RECORDINGID GetRecordingID(void);
        inline const gm::string& GetTTSAlternate(void);
        inline bool GetUseSearchPath(void);
};
inline
PlayMessage::PlayMessage(CallToken& token, const gm::string& promptname,
        const gm::string& productname, GMLANGID lang, const gm::string& tts_alternate)
        : cUseSearchPath(false), OneCallTokenMessageBase(M_CMD_PLAY, token)
{
        cPromptName = promptname.c_str();
        cProductName = productname.c_str();
        cLangID = lang;
        cTTSAlternate = tts_alternate;
        cRecordingID = INVALID_RECORDINGID;
```

```
}
inline
PlayMessage::PlayMessage(CallToken& token, RECORDINGID recordingID)
        : cUseSearchPath(false), OneCallTokenMessageBase(M_CMD_PLAY, token)
{
        cRecordingID = recordingID;
        cLangID = INVALID_GMLANGID;
}
inline
PlayMessage::PlayMessage(CallToken& token, const gm::string& promptname,
        const gm::string& tts_alternate, bool bUseSearchPath)
        : cUseSearchPath(bUseSearchPath), OneCallTokenMessageBase(M_CMD_PLAY, token)
{
        cPromptName = promptname.c_str();
        cLangID = INVALID_GMLANGID;
        cTTSAlternate = tts_alternate;
        cRecordingID = INVALID_RECORDINGID;
}
/*inline
PlayMessage::PlayMessage(CallToken& token, const gm::string& fullpath,
        const gm::string& tts_alternate)
        : cUseSearchPath(false), OneCallTokenMessageBase(M_CMD_PLAY, token)
{
        cPromptName = fullpath;
        cLangID = INVALID_GMLANGID;
        cTTSAlternate = tts_alternate;
        cRecordingID = INVALID_RECORDINGID;
}
*/
inline const gm::string&
PlayMessage::GetProductName(void)
{
        return cProductName;
}
inline const gm::string&
PlayMessage::GetPromptName(void)
{
        return cPromptName;
}
inline GMLANGID
PlayMessage::GetLangID(void)
{
        return cLangID;
}
inline RECORDINGID
PlayMessage::GetRecordingID(void)
{
        return cRecordingID;
}
inline const gm::string&
PlayMessage::GetTTSAlternate(void)
{
        return cTTSAlternate;
}
inline bool
PlayMessage::GetUseSearchPath(void)
```

```
{
        return cUseSearchPath;
}
#endif
```

Document ID: 20020826.0000

```
/*
 * PromptMgr.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.6 $
 * $Date: 2000/12/07 03:20:08 $
 */
#ifndef PROMPTMGR_H
#define PROMPTMGR_H
#include "VPPUtils/GMString"
#include "VPPUtils/GMLangID.h"
#include "VPPUtils/CriticalSection.h"
class PromptManager
{
        static const short MAXSEARCHPATH;
        static CriticalSection cCriticalSection;
        static gm::string cPromptRoot;
        static bool cRandomizerInitialized;
        static bool GenerateUniqueFilename(const gm::string& productname,
                GMLANGID langID, const gm::string& subdir, gm::string& result);
        static bool GenerateUniqueFilename(const gm::string& subdir, gm::string& result);
public:
        static void SetPromptRoot(const gm::string& promptroot);
        inline static const gm::string& GetPromptRoot(void);

        static bool BuildPromptFilename(const gm::string& productname,
                const gm::string& basename, GMLANGID langID, gm::string& result);
        static bool BuildPromptFilenameUsingSearchPath(const gm::string& searchpath,
                const gm::string& basename, gm::string& result);
        static bool BuildRecordingFilename(const gm::string& productname,
                GMLANGID langID, gm::string& result);
        static bool BuildTTSFilename(const gm::string& productname, GMLANGID langID,
                gm::string& result);
        static bool BuildTTSFilename(gm::string& result);
        static bool GenerateRandomUniqueFilename(gm::string& result);
};
inline const gm::string&
PromptManager::GetPromptRoot(void)
{
        return cPromptRoot;
}
#endif
```

```
/*
 * PromptSegment.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.7 $
 * $Date: 2000/12/07 03:20:08 $
 */
#ifndef PROMPTSEGMENT_H
#define PROMPTSEGMENT_H
class MediaChannel;
class CallToken;
class PromptSegment
{
public:
        typedef enum
        {
                ST_Undefined,
                ST_Sample,
                ST_Silence,
                ST_TTS,
    ST_DTMF
        } SegmentType;
protected:
        SegmentType cType;
public:
        inline PromptSegment(void);
        inline PromptSegment(SegmentType type);
        virtual ~PromptSegment(void);
        virtual bool Play(MediaChannel *pMediaChannel, CallToken& ct) = 0;
        inline SegmentType GetSegmentType(void);
};
inline
PromptSegment::PromptSegment(void)
{
        cType = ST_Undefined;
}
inline
PromptSegment::PromptSegment(SegmentType type)
{
        cType = type;
}
inline PromptSegment::SegmentType
PromptSegment::GetSegmentType(void)
{
        return cType;
}
#endif
```

```
/*
 * PromptSegmentMgr.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:20:08 $
 */
#ifndef PROMPTSEGMENTMGR_H
#define PROMPTSEGMENTMGR_H
#include <list>
#include "PromptSegment.h"
#pragma warning(disable:4786)
template class __declspec(dllexport) std::list<PromptSegment *>;
class __declspec(dllexport) std::list<PromptSegment *>::iterator;
class __declspec(dllexport) PromptSegmentManager
{
        std::list<PromptSegment *> cList;
        std::list<PromptSegment *>::iterator cIter;
public:
        PromptSegmentManager(void);
        ~PromptSegmentManager(void);
        void Clear(void);
        void AddSegment(PromptSegment *pSegment);
        inline PromptSegment *GetFirst(void);
        inline PromptSegment *GetNext(void);
        inline PromptSegment *GetLast(void);
};
inline PromptSegment *
PromptSegmentManager::GetFirst(void)
{
        if (cList.size())
        {
                cIter = cList.begin();
                return *cIter;
        }
        else
                return 0;
}
inline PromptSegment *
PromptSegmentManager::GetNext(void)
{
        cIter++;
        if (cIter != cList.end())
                return *cIter;
        else
                return 0;
}
inline PromptSegment *
PromptSegmentManager::GetLast(void)
{
        if (cList.size())
```

```
        {
                cIter = cList.end();
                cIter--;
                return *cIter;
        }
        else
                return 0;
}
#endif
```

```
/*
 * Property.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:21:22 $
 */
#ifndef PROPERTY_H
#define PROPERTY_H
#include "VPPUtils/GMString"
class VPPMessage;
class __declspec(dllexport) Property
{
        gm::string cName;
        gm::string cValue;
public:
        inline Property(void);
        inline Property(const gm::string& name);
        inline Property(const gm::string& name, const gm::string& value);
        Property(const gm::string& name, long value);
        inline void Setup(const gm::string& name, const gm::string& value);
        void Setup(const gm::string& name, long value);
        inline const gm::string& GetName(void);
        inline const gm::string& GetValue(void);
        long GetLongValue(void);
        void Serialize(VPPMessage *pMsg);
        void Deserialize(VPPMessage *pMsg);
};
inline void
Property::Setup(const gm::string& name, const gm::string& value)
{
        cName = name.c_str();
        cValue = value.c_str();
}
inline
Property::Property(void)
{
}
inline
Property::Property(const gm::string& name)
{
        Setup(name, 0);
}
inline
Property::Property(const gm::string& name, const gm::string& value)
{
        Setup(name, value);
}
inline const gm::string&
Property::GetName(void)
{
```

```
        return cName;
}
inline const gm::string&
Property::GetValue(void)
{
        return cValue;
}
#endif
```

Document not to be copied

```
/*
 * PropertyGottenNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:23 $
 */
#ifndef PROPERTYGOTTENNTFY_H
#define PROPERTYGOTTENNTFY_H
#include "PropertyMsg.h"
#include "VPPDefs.h"
class __declspec(dllexport) PropertyGottenNotification : public PropertyMessage
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(PropertyGottenNotification, PropertyMessage);
        inline PropertyGottenNotification(const gm::string& name, const gm::string& value);
};
inline
PropertyGottenNotification::PropertyGottenNotification(const gm::string& name,
        const gm::string& value) : PropertyMessage(M_NTFY_PROPERTYGOTTEN, name, value)
{
}
#endif
```

```
/*
 * PropertyList.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:21:23 $
 */
#ifndef PROPERTYLIST_H
#define PROPERTYLIST_H
#include <list>
#include <VPPUtils/GMString>
#include "Property.h"
#pragma warning(disable:4786)
class VPPMessage;
class __declspec(dllexport) PropertyList
{
private:
        template class __declspec(dllexport) std::list<Property *>;
        std::list<Property *> cProperties;
public:
        inline PropertyList(void);
        virtual ~PropertyList(void);
        bool RemoveProperty(const gm::string& name);
        bool SetProperty(const gm::string& name, const gm::string& value);
        bool SetProperty(const gm::string& name, long value);

        bool GetProperty(const gm::string& name, gm::string& value);
        bool GetProperty(const gm::string& name, long& value);
        inline std::list<Property *> const &GetList(void);
        void Merge(const PropertyList& other, bool EraseFirst = false);
        PropertyList& operator=(const PropertyList& other);
        void Serialize(VPPMessage *pMsg);
        void Deserialize(VPPMessage *pMsg);
};
inline
PropertyList::PropertyList(void)
{
}
inline std::list<Property *> const &
PropertyList::GetList(void)
{
        return cProperties;
}
#endif
```

```
/*
 * PropertyMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:24 $
 */
#ifndef PROPERTYMSG_H
#define PROPERTYMSG_H
#include "VPPMsg.h"
#include "Property.h"
class __declspec(dllexport) PropertyMessage: public VPPMessage
{
        Property cProperty;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);

public:
        DECLARE_CMSMESSAGE_SUBCLASS(PropertyMessage, VPPMessage);
        inline PropertyMessage(unsigned short cmdid);
        inline PropertyMessage(unsigned short cmdid,
                const gm::string& name);
        inline PropertyMessage(unsigned short cmdid,
                const gm::string& name,
                const gm::string& value);
        inline Property& GetProperty(void);
};
inline
PropertyMessage::PropertyMessage(unsigned short cmdid)
        : VPPMessage(cmdid)
{
}
inline
PropertyMessage::PropertyMessage(unsigned short cmdid, const gm::string& name)
        : VPPMessage(cmdid), cProperty(name)
{
}
inline
PropertyMessage::PropertyMessage(unsigned short cmdid, const gm::string& name,
        const gm::string& value) : VPPMessage(cmdid), cProperty(name, value)
{
}
inline Property&
PropertyMessage::GetProperty(void)
{
        return cProperty;
}
#endif
```

```
/*
 * PropertyNotGottenNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:24 $
 */
#ifndef PROPERTYNOTGOTTENNTFY_H
#define PROPERTYNOTGOTTENNTFY_H
#include "PropertyMsg.h"
#include "VPPDefs.h"
class __declspec(dllexport) PropertyNotGottenNotification : public PropertyMessage
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(PropertyNotGottenNotification, PropertyMessage);
        inline PropertyNotGottenNotification(const gm::string& name, const gm::string& value);
};
inline
PropertyNotGottenNotification::PropertyNotGottenNotification(const gm::string& name,
        const gm::string& value) : PropertyMessage(M_NTFY_PROPERTYNOTGOTTEN, name, value)
{
}
#endif
```

```
/*
 * PropertyNotSetNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:24 $
 */
#ifndef PROPERTYNOTSETNTFY_H
#define PROPERTYNOTSETNTFY_H
#include "PropertyMsg.h"
#include "VPPDefs.h"
class __declspec(dllexport) PropertyNotSetNotification : public PropertyMessage
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(PropertyNotSetNotification, PropertyMessage);
        inline PropertyNotSetNotification(const gm::string& name, const gm::string& value);
};
inline
PropertyNotSetNotification::PropertyNotSetNotification(const gm::string& name, const gm::string& value)
        : PropertyMessage(M_NTFY_PROPERTYNOTSET, name, value)
{
}
#endif
```

```
/*
 * PropertySetNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:25 $
 */
#ifndef PROPERTYSETNTFY_H
#define PROPERTYSETNTFY_H
#include "PropertyMsg.h"
#include "VPPDefs.h"
class __declspec(dllexport) PropertySetNotification : public PropertyMessage
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(PropertySetNotification, PropertyMessage);
        inline PropertySetNotification(const gm::string& name, const gm::string& value);
};
inline
PropertySetNotification::PropertySetNotification(const gm::string& name, const gm::string& value)
        : PropertyMessage(M_NTFY_PROPERTYSET, name, value)
{
}
#endif
```

```
/*
 * QueryDynamicGrammarContentsMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:25 $
 */
#ifndef QUERYDYNAMICGRAMMARCONTENTSMSG_H
#define QUERYDYNAMICGRAMMARCONTENTSMSG_H
#include "DynamicGrammarPhraseMsg.h"
class __declspec(dllexport) QueryDynamicGrammarContentsMessage : public DynamicGrammarPhraseMessage
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(QueryDynamicGrammarContentsMessage,
DynamicGrammarPhraseMessage);
        inline QueryDynamicGrammarContentsMessage(CallToken& token, const gm::string& db_key,
                const gm::string &phrase_id);
};
inline
QueryDynamicGrammarContentsMessage::QueryDynamicGrammarContentsMessage(CallToken& token,
        const gm::string& db_key, const gm::string &phrase_id)
        : DynamicGrammarPhraseMessage(M_CMD_QUERYDYNAMICGRAMMARCONTENTS, token, db_key,
phrase_id)
{
}
#endif
```

```
/*
 * QueryDynamicGrammarContentsResultNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:25 $
 */
#ifndef QUERYDYNAMICGRAMMARCONTENTSRESULTNTFY_H
#define QUERYDYNAMICGRAMMARCONTENTSRESULTNTFY_H
#include "VPPDefs.h"
#include "DynamicGrammarPhraseResultNtfy.h"
class __declspec(dllexport) QueryDynamicGrammarContentsResultNotification : public
DynamicGrammarPhraseResultNotification
{
        gm::string cPhraseContents;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(QueryDynamicGrammarContentsResultNotification,
DynamicGrammarPhraseResultNotification);
        inline QueryDynamicGrammarContentsResultNotification(CallToken& ct, const gm::string& db_key, Result
result,
                const gm::string& phrase_id, const gm::string& phrase_contents);
        inline gm::string& GetPhraseContents(void);
};
inline
QueryDynamicGrammarContentsResultNotification::QueryDynamicGrammarContentsResultNotification(CallToken&
callToken,
         const gm::string& db_key, DynamicGrammarResultNotification::Result result, const gm::string& phrase_id,
         const gm::string& phrase_contents)
        :
DynamicGrammarPhraseResultNotification(M_NTFY_QUERYDYNAMICGRAMMARCONTENTSRESULT,
callToken, db_key, result, phrase_id),
        cPhraseContents(phrase_contents.c_str())
{
}
inline gm::string&
QueryDynamicGrammarContentsResultNotification::GetPhraseContents(void)
{
        return cPhraseContents;
}
#endif
```

```
/*
 * Queue.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.10 $
 * $Date: 2000/12/07 03:22:24 $
 */
#ifndef _QUEUE_H
#define _QUEUE_H
#include "VPPUtils/Monitor.h"
#include "VPPUtils/Synchronized.h"
/**
 *
 * Circular FIFO Queue
 *
 */
template<class T>
class __declspec(dllexport) Queue : public Monitor {
  typedef T T_PTR;
public:
  Queue() : head(0), tail(0), qsize(4) {
    elementData = new T_PTR[qsize];
  };
  Queue(int num) : head(0), tail(0), qsize(num) {
    elementData = new T_PTR[qsize];
  };
  virtual ~Queue() {
    delete [] elementData;
  }
  T peek() {
    Synchronized sync(this);
    if (head == tail) {
      return NULL;
    }
    return elementData[tail];
  };
  virtual T pop() {
    Synchronized sync(this);
    if (head == tail) {
      return NULL;
    }
    T o = elementData[tail];
    elementData[tail++] = NULL;
    if (tail == qsize) {
      tail = 0;
    }
    return o;
  };
  virtual void push(T o) {
    Synchronized sync(this);
    elementData[head++] = o;
```

```
    if (head == qsize) {
      head = 0;
    }
    if (head == tail) {
      // time to grow
      grow();
    }
};
int size() {
    Synchronized sync(this);
    if (tail <= head) {
      return head - tail;
    } else {
      return qsize - (tail - head);
    }
};
T elementAt(int i) {
    Synchronized sync(this);
    if (i >= size()) {
      return NULL;
    }
    int pos = tail + i;
    if (pos >= qsize) {
      pos -= qsize;
    }
    return elementData[pos];
}
bool contains(T o) {
    return indexOf(o) >= 0;
};
bool remove(T o) {
    Synchronized sync(this);
    int index = indexOf(o);
    if (index >= tail) {
      int to = tail +1;
      if (to < qsize) {
        arraycopy(elementData, tail,
              elementData, to,
              index - tail);
        elementData[tail] = NULL;
        tail = to;
      } else if (to == qsize) {
        elementData[tail] = NULL;
        tail = 0;
      } else {
        return false;
      }
    } else if (index>=0) {
      int from = index + 1;
      // (index < tail) means that "from" is not at qsize.
      arraycopy(elementData, from,
            elementData, index,
            head - index);
      elementData[head] = NULL;
      head --;
      if (head <0) {
```

```
        head = qsize - 1;
      }
    } else {
      return false;
    }
    return true;
};
int indexOf(T o) {
    Synchronized sync(this);
    if (tail <= head) {
      for (int i=tail; i < head; i++) {
        if (o == elementData[i]) {
          return i;
        }
      }
    } else {
      for (int i=0; i < head; i++) {
        if (o == elementData[i]) {
          return i;
        }
      }
      for (i=tail; i < qsize; i++) {
        if (o == elementData[i]) {
          return i;
        }
      }
    }
    return -1;
};

void grow() {
    // REVISIT: why need to allocate objects for old items?
        // I might be sleeping...
    int oldCapacity = qsize;
    T_PTR* oldData = elementData;
    qsize = (oldCapacity * 2);
    elementData = new T_PTR[qsize];
    int n = 0;
    arraycopy(oldData, tail,
            elementData, 0,
            oldCapacity - tail);
    arraycopy(oldData, 0,
            elementData, oldCapacity - tail,
            head);
    delete [] oldData;
    tail = 0; head = oldCapacity;
};
static void arraycopy(T_PTR* src, int src_pos, T_PTR* to, int to_pos, int length) {
    src = & (src[src_pos]);
    to = & (to[to_pos]);
    if (src == to) {
      return;
    } else if (src < to) {
      for (int i=length-1; i>=0; i--) {
        to[i] = src[i];
      }
```

```
    } else {
      memcpy(to, src, length * sizeof(T_PTR));
    }
  };
/*
  String toString() {
    StringBuffer buf = new StringBuffer();
    buf.append('[');
    for (int i=0; i<elementData.length; i++) {
      buf.append(elementData[i]);
      buf.append(',');
    }
    buf.append(']');
    return buf.toString();
  };
*/
private:
  T_PTR* elementData;
  int qsize;
  int head, tail;
};
template<class T>
class __declspec(dllexport) BlockingQueue : public Queue<T> {
  bool running;
public:
  BlockingQueue() : running(true), Queue<T>() {}
  void stop() {
    Synchronized sync(this);
    running = false;
    sync.notifyAll();
  }
  virtual T pop() {
    Synchronized sync(this);
    T o = NULL;
    while (running && (o = Queue<T>::pop()) == NULL){
      sync.wait();
    }
    return o;
  }

  virtual void push(T o) {
    Synchronized sync(this);
    Queue<T>::push(o);
    sync.notify();
  }
};
#endif
```

```
/*
 * RangeParser.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:20:09 $
 */
#ifndef RANGEPARSER_H
#define RANGEPARSER_H
#include "VPPUtils/GMString"
class __declspec(dllexport) RangeParser
{
        unsigned short cMaxValue;
        unsigned char *cBits;
        unsigned short Parse(const gm::string& str, bool doit);
        void SetBit(unsigned short ndx);
public:
        RangeParser(const gm::string& str);
        ~RangeParser(void);
        inline unsigned short GetMaxValue(void);
        bool IsSet(unsigned short ndx);
};
inline unsigned short
RangeParser::GetMaxValue(void)
{
        return cMaxValue;
}
#endif
```

```
/*
 * RecognitionDoneNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:21:25 $
 */
#ifndef RECOGNITIONDONENTFY_H
#define RECOGNITIONDONENTFY_H
#include "VPPDefs.h"
#include "OneCallTokenMsgBase.h"
class RecognitionResults;
class __declspec(dllexport) RecognitionDoneNotification : public OneCallTokenMessageBase
{
public:
        typedef enum
        {
                RS_SuccessfulSpeech,
                RS_SuccessfulDTMF,
                RS_NoSpeech,
                RS_TooMuchSpeech,
                RS_CallDropped,
                RS_Error
        } RecognitionStatus;
private:
        RecognitionStatus cStatus;
        RecognitionResults *cResults;
        bool cOwned;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(RecognitionDoneNotification, OneCallTokenMessageBase);
        inline RecognitionDoneNotification(CallToken& ct, RecognitionStatus status,
                RecognitionResults *pResults);
        virtual ~RecognitionDoneNotification(void);
        inline RecognitionResults *GetResults(void);
        inline RecognitionStatus GetStatus(void);
};
inline
RecognitionDoneNotification::RecognitionDoneNotification(CallToken& ct,
        RecognitionStatus status, RecognitionResults *pResults)
        : OneCallTokenMessageBase(M_NTFY_RECOGNITIONDONE, ct), cResults(pResults), cStatus(status)
{
        cOwned = false;
}
inline RecognitionResults *
RecognitionDoneNotification::GetResults(void)
{
        return cResults;
}
```

```
inline RecognitionDoneNotification::RecognitionStatus
RecognitionDoneNotification::GetStatus(void)
{
        return cStatus;
}
#endif
```

Document #

```
/*
 * RecognitionResult.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.9 $
 * $Date: 2000/12/07 03:21:26 $
 */
#ifndef RECOGNITIONRESULT_H
#define RECOGNITIONRESULT_H
#include "NLInterpretation.h"
#include <list>
#pragma warning(disable:4786)
#pragma warning(disable:4660)
class VPPMessage;
class __declspec(dllexport) RecognitionResult
{
        gm::string cResult;
        float cConfidence;
        template class __declspec(dllexport) std::list<NLInterpretation *>;
        std::list<NLInterpretation *> cNLInterpretations;
public:
        RecognitionResult(void);
        ~RecognitionResult(void);
        inline const gm::string& GetResult(void);
        inline float GetConfidence(void);
        unsigned short GetNumNLInterpretations(void);
        NLInterpretation *GetNLInterpretation(int index);
        inline void SetResult(const gm::string& result);
        inline void SetConfidence(float confidence);
        void AddNLInterpretation(NLInterpretation *pNLInterpretation);
        void Serialize(VPPMessage *);
        void Deserialize(VPPMessage *);
};
inline void
RecognitionResult::SetResult(const gm::string& result)
{
        cResult = result.c_str();
}
inline void
RecognitionResult::SetConfidence(float confidence)
{
        cConfidence = confidence;
}

inline const gm::string&
RecognitionResult::GetResult(void)
{
        return cResult;
}
inline float
RecognitionResult::GetConfidence(void)
```

```
{
        return cConfidence;
}
#endif
```

Document ID: 20020826

```
/*
 * RecognitionResults.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:21:26 $
 */
#ifndef RECOGNITIONRESULTS_H
#define RECOGNITIONRESULTS_H
#include "RecognitionResult.h"
#include <list>
#pragma warning(disable:4786)
class VPPMessage;
class __declspec(dllexport) RecognitionResults
{
        template class __declspec(dllexport) std::list<RecognitionResult *>;
        std::list<RecognitionResult *> cResults;
public:
        RecognitionResults(void);
        ~RecognitionResults(void);
        unsigned short GetNumResults(void);
        RecognitionResult *GetResult(int index);
        void AddResult(RecognitionResult *result);
        void Serialize(VPPMessage *);
        void Deserialize(VPPMessage *);
};
#endif
```

```
/*
 * RecognitionStartedNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:27 $
 */
#ifndef RECOGNITIONSTARTEDNTFY_H
#define RECOGNITIONSTARTEDNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) RecognitionStartedNotification : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(RecognitionStartedNotification,
                OneCallTokenMessageBase);
        inline RecognitionStartedNotification(CallToken& token);
};
inline
RecognitionStartedNotification::RecognitionStartedNotification(CallToken& token)
        : OneCallTokenMessageBase(M_NTFY_RECOGNITIONSTARTED, token)
{
}
#endif
```

```
/*
 * RecognizeStringMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:21:27 $
 */
#ifndef RECOGNIZESTRINGMSG_H
#define RECOGNIZESTRINGMSG_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
#include "VPPUtils/GMString"
class __declspec(dllexport) RecognizeStringMessage :
        public OneCallTokenMessageBase
{
        gm::string cGrammar;
        gm::string cStr;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(RecognizeStringMessage, OneCallTokenMessageBase);
        inline RecognizeStringMessage(CallToken& token, const gm::string& grammar,
                const gm::string& str);
        inline const gm::string& GetGrammar(void);
        inline const gm::string& GetString(void);
};
inline
RecognizeStringMessage::RecognizeStringMessage(CallToken& token, const gm::string& grammar,
        const gm::string& str) : OneCallTokenMessageBase(M_CMD_RECOGNIZESTRING, token)
{
        cGrammar = grammar.c_str();
        cStr = str.c_str();
}
inline const gm::string&
RecognizeStringMessage::GetGrammar(void)
{
        return cGrammar;
}
inline const gm::string&
RecognizeStringMessage::GetString(void)
{
        return cStr;
}
#endif
```

```
/*
 * RecordingDoneNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:21:28 $
 */
#ifndef RECORDINGDONENTFY_H
#define RECORDINGDONENTFY_H
#include "OneCallTokenMsgBase.h"
class __declspec(dllexport) RecordingDoneNotification : public OneCallTokenMessageBase
{
public:
   typedef enum
   {
      RDR_NoSpeech,
      RDR_EndOfSpeech,
      RDR_TooMuchSpeech,
      RDR_Error
   } RecordingDoneReason;
private:
   RecordingDoneReason cReason;
protected:
         virtual void Serialize(void);
         virtual void Deserialize(void);
public:
         DECLARE_CMSMESSAGE_SUBCLASS(RecordingDoneNotification,
               OneCallTokenMessageBase);
         inline RecordingDoneNotification(CallToken& token);
         inline RecordingDoneReason GetReason(void);
};
#endif
```

```
/*
 * RecordingHandedOffNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:28 $
 */
#ifndef RECORDINGHANDEDOFFNTFY_H
#define RECORDINGHANDEDOFFNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) RecordingHandedOffNotification
        : public OneCallTokenMessageBase
{
        RECORDINGID cID;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(RecordingHandedOffNotification,
                OneCallTokenMessageBase);
        inline RecordingHandedOffNotification(CallToken& token,
                RECORDINGID recordingID);
        inline RECORDINGID GetRecordingID(void);
};
inline
RecordingHandedOffNotification::RecordingHandedOffNotification(CallToken& token,
        RECORDINGID recordingID)
        : OneCallTokenMessageBase(M_NTFY_RECORDINGHANDEDOFF, token)
{
        cID = recordingID;
}
inline RECORDINGID
RecordingHandedOffNotification::GetRecordingID(void)
{
        return cID;
}
#endif
```

```
/*
 * RecordingHandler.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:20:09 $
 */
#ifndef RECORDINGHANDLER_H
#define RECORDINGHANDLER_H
#include "VPPUtils/GMString"
class __declspec(dllexport) RecordingHandler
{
public:
        virtual ~RecordingHandler(void);
        virtual bool Initialize(void) = 0;
        virtual const gm::string& GetServiceName(void) = 0;
        virtual bool HandOff(const gm::string& filename, void *data,
                unsigned long datalen) = 0;
};
typedef RecordingHandler *(*CREATERECORDINGHANDLERFUNC)(void);
extern "C" __declspec(dllexport) RecordingHandler *CreateRecordingHandler(void);
#endif
```

```
/*
 * RecordingMgr.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:20:09 $
 */
#ifndef RECORDINGMGR_H
#define RECORDINGMGR_H
#include "VPPMsg/VPPDefs.h"
#include <list>
#include "VPPUtils/GMString"
#include "VPPUtils/CriticalSection.h"
#pragma warning(disable:4786)
class __declspec(dllexport) RecordingManager
{
        struct RecordingNode
        {
                RECORDINGID cRecordingID;
                gm::string cFilename;
        };

        static CriticalSection cCriticalSection;
        static RECORDINGID cNextRecordingID;
        template class __declspec(dllexport) std::list<RecordingNode *>;
        std::list<RecordingNode *> cRecordings;
        bool Remove(RecordingNode *);
public:
        inline RecordingManager(void);
        ~RecordingManager(void);
        RECORDINGID Add(gm::string& filename);

        bool Remove(gm::string& filename);
        bool Remove(RECORDINGID recordingID);

        RECORDINGID Lookup(gm::string& filename);
        bool Lookup(RECORDINGID recordingID, gm::string& filename);
        int GetNumEntries(void);
        bool GetFilename(int index, gm::string& filename);
};
inline
RecordingManager::RecordingManager(void)
{
}
#endif
```

```
/*
 * RecordingNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:21:28 $
 */
#ifndef RECORDINGNTFY_H
#define RECORDINGNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
#include "VPPUtils/GMString"
class __declspec(dllexport) RecordingNotification : public OneCallTokenMessageBase
{
        RECORDINGID cID;
        gm::string      cRecordingFilename;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(RecordingNotification,
                OneCallTokenMessageBase);
        inline RecordingNotification(CallToken& token, RECORDINGID recordingID, const gm::string& filename);
        inline RECORDINGID GetRecordingID(void);
        const gm::string& GetRecordingFilename(void);
};
inline
RecordingNotification::RecordingNotification(CallToken& token,
        RECORDINGID recordingID, const gm::string& filename) :
OneCallTokenMessageBase(M_NTFY_RECORDING, token)
{
        cID = recordingID;
        cRecordingFilename = filename.c_str();
}
inline RECORDINGID
RecordingNotification::GetRecordingID(void)
{
        return cID;
}
#endif
```

```
/*
 * RecordMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.6 $
 * $Date: 2000/12/07 03:21:27 $
 */
#ifndef RECORDMSG_H
#define RECORDMSG_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
#include "VPPUtils/GMString"
//#include "VPPUtils/GMLangID.h"
class __declspec(dllexport) RecordMessage : public OneCallTokenMessageBase
{
        gm::string cFileName;
   unsigned long cBosTimeout, cTmsTimeout, cEosTimeout;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);

public:
        DECLARE_CMSMESSAGE_SUBCLASS(RecordMessage, OneCallTokenMessageBase);
        inline RecordMessage(CallToken& token, const gm::string& filename,
     unsigned long bos_timeout, unsigned long tms_timeout, unsigned long eos_timeout);
        inline const gm::string& GetFileName(void);
   inline unsigned long GetBosTimeout(void);
   inline unsigned long GetTmsTimeout(void);
   inline unsigned long GetEosTimeout(void);
};
inline
RecordMessage::RecordMessage(CallToken& token, const gm::string& filename,
                unsigned long bos_timeout, unsigned long tms_timeout, unsigned long eos_timeout)
: OneCallTokenMessageBase(M_CMD_RECORD, token),
 cBosTimeout(bos_timeout), cTmsTimeout(tms_timeout), cEosTimeout(eos_timeout)
{
        cFileName = filename.c_str();
}
inline const gm::string&
RecordMessage::GetFileName(void)
{
        return cFileName;
}
inline unsigned long
RecordMessage::GetBosTimeout(void)
{
   return cBosTimeout;
}
inline unsigned long
RecordMessage::GetTmsTimeout(void)
{
```

```
  return cTmsTimeout;
}
inline unsigned long
RecordMessage::GetEosTimeout(void)
{
  return cEosTimeout;
}
#endif
```

Document ID: 0.0.2.0.0.0

```
/*
 * RegistryMgr.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:22:25 $
 */
#ifndef REGISTRYMGR_H
#define REGISTRYMGR_H
#include "GMString"
class __declspec(dllexport) RegistryMgr
{
public:
        static bool GetString(const char *parent, const char *key, gm::string& ret);
};
#endif
```

Doc ID: 5.6.2.2.0

```
/*
 * RegularExpression.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:33:44 $
 */
#ifndef REGULAREXPRESSION_H
#define REGULAREXPRESSION_H
struct re_pattern_buffer;
class __declspec(dllexport) RegularExpression
{
        struct re_pattern_buffer *cPatternBuffer;
        void freeStrings(void);
public:
        RegularExpression(const char *str = 0);
        ~RegularExpression(void);
        const char *setString(const char *str);
        int match(const char *str, int startpos);
        int search(const char *str, int startpos, int range);
};
#endif
```

```
/*
 * RejectCallMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:29 $
 */
#ifndef REJECTCALLMSG_H
#define REJECTCALLMSG_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) RejectCallMessage : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(RejectCallMessage, OneCallTokenMessageBase);
        inline RejectCallMessage(CallToken& token);
};
inline
RejectCallMessage::RejectCallMessage(CallToken& token)
        : OneCallTokenMessageBase(M_CMD_REJECTCALL, token)
{
}
#endif
```

```
/*
 * RemovePhraseFromDynamicGrammarMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:29 $
 */
#ifndef REMOVEPHRASEFROMDYNAMICGRAMMARMSG_H
#define REMOVEPHRASEFROMDYNAMICGRAMMARMSG_H
#include "DynamicGrammarPhraseMsg.h"
class __declspec(dllexport) RemovePhraseFromDynamicGrammarMessage : public DynamicGrammarPhraseMessage
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(RemovePhraseFromDynamicGrammarMessage,
DynamicGrammarPhraseMessage);
        inline RemovePhraseFromDynamicGrammarMessage(CallToken& token, const gm::string& db_key,
                const gm::string &phrase_id);
};
inline
RemovePhraseFromDynamicGrammarMessage::RemovePhraseFromDynamicGrammarMessage(CallToken& token,
        const gm::string& db_key, const gm::string &phrase_id)
        : DynamicGrammarPhraseMessage(M_CMD_REMOVEPHRASEFROMDYNAMICGRAMMAR, token,
db_key, phrase_id)
{
}
#endif
```

```
/*
 * RemovePhraseFromDynamicGrammarResultNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:29 $
 */
#ifndef REMOVEPHRASEFROMDYNAMICGRAMMARRESULTNTFY_H
#define REMOVEPHRASEFROMDYNAMICGRAMMARRESULTNTFY_H
#include "DynamicGrammarPhraseResultNtfy.h"
class __declspec(dllexport) RemovePhraseFromDynamicGrammarResultNotification : public
DynamicGrammarPhraseResultNotification
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(RemovePhraseFromDynamicGrammarResultNotification,
DynamicGrammarPhraseResultNotification);
        inline RemovePhraseFromDynamicGrammarResultNotification(CallToken& ct, const gm::string& db_key,
Result result,
                const gm::string& phrase_id);
};
inline
RemovePhraseFromDynamicGrammarResultNotification::RemovePhraseFromDynamicGrammarResultNotification(C
allToken& callToken,
        const gm::string& db_key, DynamicGrammarResultNotification::Result result, const gm::string& phrase_id)
        :
DynamicGrammarPhraseResultNotification(M_NTFY_REMOVEPHRASEFROMDYNAMICGRAMMARRESULT,
callToken, db_key, result, phrase_id)
{
}
#endif
```

```
/*
 * RemoveRecordingMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:30 $
 */
#ifndef REMOVERECORDINGMSG_H
#define REMOVERECORDINGMSG_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) RemoveRecordingMessage : public OneCallTokenMessageBase
{
        RECORDINGID cID;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(RemoveRecordingMessage,
                OneCallTokenMessageBase);
        inline RemoveRecordingMessage(CallToken& token, RECORDINGID recordingID);
        inline RECORDINGID GetRecordingID(void);
};
inline
RemoveRecordingMessage::RemoveRecordingMessage(CallToken& token,
        RECORDINGID recordingID)
        : OneCallTokenMessageBase(M_CMD_REMOVERECORDING, token)
{
        cID = recordingID;
}
inline RECORDINGID
RemoveRecordingMessage::GetRecordingID(void)
{
        return cID;
}
#endif
```

```
/*
 * RequestCancelledNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:21:30 $
 */
#ifndef REQUESTCANCELLEDNTFY_H
#define REQUESTCANCELLEDNTFY_H
#include "VPPDefs.h"
#include "VPPMsg.h"
#include "RequestToken.h"
class __declspec(dllexport) RequestCancelledNotification : public VPPMessage
{
        RequestToken cRequest;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);

public:
        DECLARE_CMSMESSAGE_SUBCLASS(RequestCancelledNotification, VPPMessage);
        inline RequestCancelledNotification(RequestToken& request);
        inline RequestToken& GetRequestToken(void);
};
inline
RequestCancelledNotification::RequestCancelledNotification(RequestToken& request)
        : VPPMessage(M_NTFY_REQUESTCANCELLED)
{
        cRequest = request;
}
inline RequestToken&
RequestCancelledNotification::GetRequestToken(void)
{
        return cRequest;
}
#endif
```

```
/*
 * RequestToken.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:21:31 $
 */
#ifndef REQUESTTOKEN_H
#define REQUESTTOKEN_H
#include "VPPDefs.h"
#if 0
#include "ClientID.h"
#endif
class VPPMessage;
class __declspec(dllexport) RequestToken
{
#if 0
        ClientID cClientID;
        VirtualMachineID cMTSVirtualMachineID;
#endif
        REQUESTID cRequestID;
public:
        inline RequestToken(void);
        inline RequestToken(REQUESTID requestID);
#if 0
        inline RequestToken(ClientID& clientID, REQUESTID requestID);
        inline void SetClientID(ClientID& clientID);
        inline ClientID& GetClientID(void);
#endif
        inline void SetRequestID(REQUESTID requestID);
        inline REQUESTID GetRequestID(void);
#if 0
        inline void SetMTSVirtualMachineID(VirtualMachineID& vmid);
        inline VirtualMachineID& GetMTSVirtualMachineID(void);
#endif
        inline void Invalidate(void);
        inline bool IsValid(void);
        inline bool operator==(const RequestToken& other);
        inline bool operator!=(const RequestToken& other);
        void Serialize(VPPMessage *);
        void Deserialize(VPPMessage *);
};
inline void
RequestToken::Invalidate(void)
{
#if 0
        cClientID.Invalidate();
#endif
        cRequestID = INVALID_REQUESTID;
}
inline
```

```
RequestToken::RequestToken(void)
{
        Invalidate();
}
inline
RequestToken::RequestToken(REQUESTID requestID)
{
  cRequestID = requestID;
}
#if 0
inline
RequestToken::RequestToken(ClientID& clientID, REQUESTID requestID)
{
        cClientID = clientID;
        cRequestID = requestID;
}
inline void
RequestToken::SetClientID(ClientID& clientID)
{
        cClientID = clientID;
}
inline ClientID&
RequestToken::GetClientID(void)
{
        return cClientID;
}
#endif
inline void
RequestToken::SetRequestID(REQUESTID requestID)
{
        cRequestID = requestID;
}
inline REQUESTID
RequestToken::GetRequestID(void)
{
        return cRequestID;
}
inline bool
RequestToken::IsValid(void)
{
#if 0
        return (bool)(
                cClientID.IsValid() &&
                (cRequestID != INVALID_REQUESTID));
#else
        return (bool)(cRequestID != INVALID_REQUESTID);
#endif
}
inline bool
RequestToken::operator==(const RequestToken& other)
{
#if 0
        return (bool)(
                (cClientID == other.cClientID) &&
                (cRequestID == other.cRequestID));
#else
```

```
    return (bool)(cRequestID == other.cRequestID);
#endif
}
inline bool
RequestToken::operator!=(const RequestToken& other)
{
        return !operator==(other);
}
#if 0
inline void
RequestToken::SetMTSVirtualMachineID(VirtualMachineID& vmid)
{
        cMTSVirtualMachineID = vmid;
}
inline VirtualMachineID&
RequestToken::GetMTSVirtualMachineID(void)
{
        return cMTSVirtualMachineID;
}
#endif
#endif
```

Document ID: 1.0.0.0

```
/*
 * RequestTokenMsgBase.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:31 $
 */
#ifndef REQUESTTOKENMESSAGEBASE_H
#define REQUESTTOKENMESSAGEBASE_H
#include "VPPMsg.h"
#include "RequestToken.h"
class __declspec(dllexport) RequestTokenMessageBase : public VPPMessage
{
protected:
        RequestToken cRequestToken;
        virtual void Serialize(void);
        virtual void Deserialize(void);
        inline RequestTokenMessageBase(unsigned short cmdid);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(RequestTokenMessageBase, VPPMessage);
        inline RequestTokenMessageBase(unsigned short cmdid, RequestToken& requestToken);
        inline RequestToken& GetRequestToken(void);
#if 0
        virtual VirtualMachineID& GetMTSVirtualMachineID(void);
#endif
};
inline
RequestTokenMessageBase::RequestTokenMessageBase(unsigned short cmdid)
        : VPPMessage(cmdid)
{
}
inline
RequestTokenMessageBase::RequestTokenMessageBase(unsigned short cmdid,
        RequestToken& requestToken) : VPPMessage(cmdid)
{
        cRequestToken = requestToken;
}
inline RequestToken&
RequestTokenMessageBase::GetRequestToken(void)
{
        return cRequestToken;
}
#endif
```

```
/*
 * SampleListener.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:20:10 $
 */
#ifndef SAMPLELISTENER_H
#define SAMPLELISTENER_H
class __declspec(dllexport) SampleListener
{
#ifdef ENABLE_PERF_MEASURE
protected:
        unsigned long cSamplesTime;
#endif
public:
        virtual void NotifySample(const unsigned char *pSample, unsigned int numBytes) = 0;
        virtual void NotifyEndOfData(int reason) = 0;
#ifdef ENABLE_PERF_MEASURE
        inline SampleListener(void);
        inline void SetSamplesTime(unsigned long time);
        inline unsigned long GetSamplesTime(void);
#endif
};
#ifdef ENABLE_PERF_MEASURE
inline
SampleListener::SampleListener(void)
: cSamplesTime(0L)
{
}
inline void
SampleListener::SetSamplesTime(unsigned long time)
{
        cSamplesTime = time;
}
inline unsigned long
SampleListener::GetSamplesTime(void)
{
        return cSamplesTime;
}
#endif
#endif
```

```
/*
 * SampleSegment.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.8 $
 * $Date: 2000/12/07 03:20:10 $
 */
#ifndef SAMPLESEGMENT_H
#define SAMPLESEGMENT_H
#include "PromptSegment.h"
#include "VPPUtils/GMString"
class SampleSegment : public PromptSegment
{
        gm::string cFilename;
        bool cIsTTS;
public:
        inline SampleSegment(const gm::string& filename, bool IsTTS);
        virtual ~SampleSegment(void);
        virtual bool Play(MediaChannel *pMediaChannel,
                CallToken& ct);
};

inline
SampleSegment::SampleSegment(const gm::string& filename, bool IsTTS)
        : PromptSegment(PromptSegment::ST_Sample)
{
        cFilename = filename.c_str();
        cIsTTS = IsTTS;
}
#endif
```

```
/*
 * SampleTransmitter.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:20:10 $
 */
#ifndef SAMPLETRANSMITTER_H
#define SAMPLETRANSMITTER_H
class SampleListener;
class __declspec(dllexport) SampleTransmitter
{

public:
        virtual bool StartTransmission(void) = 0;
        virtual bool StopTransmission(void) = 0;
        virtual bool SetSampleListener(SampleListener *pListener) = 0;
};
#endif
```

```
/*
 * SCChannel.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.6 $
 * $Date: 2000/12/07 03:20:09 $
 */
#ifndef SCCHANNEL_H
#define SCCHANNEL_H
#include "Channel.h"
#pragma warning(disable:4786)
class __declspec(dllexport) SCChannel : public Channel
{
protected:
        SCChannel *cChannelImListeningTo;
        template class __declspec(dllexport) std::list<SCChannel *>;
        std::list<SCChannel *> cChannelsListeningToMe;
        unsigned short cTxTimeslot;
        virtual unsigned long ListenTo(unsigned short timeslot) = 0;
        inline void NotifyListeningChannel(SCChannel *pListeningChannel);
        inline void NotifyChannelNoLongerListening(SCChannel *pChannel);

public:
        inline SCChannel(ServiceProvider *pServiceProvider, ChannelType type,
                const char *logType, int id, bool RequiresSettlement = false);
        virtual ~SCChannel(void);
        unsigned long ListenTo(SCChannel *pOtherChannel);
        inline SCChannel *GetChannelImListeningTo(void);
        inline std::list<SCChannel *>& GetChannelsListeningToMe(void);
};
inline
SCChannel::SCChannel(ServiceProvider *pServiceProvider, Channel::ChannelType type,
        const char *logType, int id, bool RequiresSettlement) : Channel(pServiceProvider, type, logType, id,
RequiresSettlement)
{
        cChannelImListeningTo = 0;
}
inline SCChannel *
SCChannel::GetChannelImListeningTo(void)
{
        return cChannelImListeningTo;
}
inline std::list<SCChannel *>&
SCChannel::GetChannelsListeningToMe(void)
{
        return cChannelsListeningToMe;
}
#ifdef LATER
inline bool
SCChannel::BelongsToSession(void)
{
```

```
        return (bool)(cSession != 0);
}
#endif
#endif
```

Oracle Exhibit 1005

```
/*
 * SecureHash.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:22:25 $
 */
#ifndef  _SECURE_HASH_H
#define  _SECURE_HASH_H
#include "GMString"
/**
 */
class __declspec(dllexport) SecureHash {
public:
   static gm::string hash(gm::string &str);
private:
   static void toPrintableString(gm::string &buf, unsigned long v);
};
#endif
```

```
/*
 * ServiceProvider.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.17 $
 * $Date: 2000/12/07 03:20:11 $
 */
#ifndef SERVICEPROVIDER_H
#define SERVICEPROVIDER_H
#include "VPPUtils/CriticalSection.h"
#include "VPPUtils/GMString"
#include "VPPUtils/IDLog.h"
#include "VPPMsg/VPPMsg.h"
#include <list>
#pragma warning(disable:4786)
#pragma warning(disable:4660)
class MTSServer;
class PropertyList;
class TelephonyChannel;
class CallToken;
class Channel;
class __declspec(dllexport) ServiceProvider : public IDLog
{
protected:
        template class __declspec(dllexport) std::list<Channel *>;
        unsigned short cFailureNotificationID;
        CriticalSection cChannelListLock;
        std::list<Channel *> cChannels;
        MTSServer *cServer;
   bool shutdown;
        bool SetDefaultProperty(const gm::string& name, const gm::string& value);
        bool SetDefaultProperty(const gm::string& name, long value);
   void Shutdown();
public:
        inline ServiceProvider(MTSServer *pServer, unsigned short FailureNotificationID, const char *logType);
        virtual ~ServiceProvider(void);
        virtual bool Setup(void) = 0;
        virtual bool PostSetup(gm::string registryParent);
        virtual bool Teardown(void) = 0;
        void AddChannel(Channel *);
        inline MTSServer *GetMTSServer(void);
        inline std::list<Channel *>& GetChannels(void);
        inline std::list<Channel *>::size_type GetNumChannels(void);
        Channel *FindChannelForCallToken(CallToken& callToken);
};
inline
ServiceProvider::ServiceProvider(MTSServer *pServer,
                                 unsigned short FailureNotificationID, const char *logType) :
                                 IDLog(logType)
{
        cServer = pServer;
```

```
        cFailureNotificationID = FailureNotificationID;
    shutdown = false;
}
inline MTSServer *
ServiceProvider::GetMTSServer(void)
{
        return cServer;
}
inline std::list<Channel *>&
ServiceProvider::GetChannels(void)
{
        return cChannels;
}
inline std::list<Channel *>::size_type
ServiceProvider::GetNumChannels(void)
{
        return cChannels.size();
}
#endif
```

```
/*
 * SessionLogger.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.8 $
 * $Date: 2000/12/07 03:20:11 $
 */
#ifndef SESSIONLOGGER_H
#define SESSIONLOGGER_H
#include "VPPUtils/GMString"
#include "TelephonyChannel.h"
class Property;
class SessionNode;
class __declspec(dllexport) SessionLogger
{
        friend class SessionMgr;
        FILE *cFile;
        unsigned long cSessionBeginTime;
        unsigned long cStateBeginTime;
        bool cInState;
        bool cLogDirCreated;
        gm::string cLogRoot;
        gm::string cLogPath;
        gm::string cLogFile;
        gm::string cIDString;
        SessionNode *cSessionNode;
        static char *cMonths[12];
        void GetNiceTime(gm::string& str);
        bool OpenFile(const gm::string& fullpath);
        bool CreateLogPath(void);
        void CreateLogPathname(void);
        bool CreateLogDirectory(void);
public:
        SessionLogger(SessionNode *pSessionNode, TelephonyChannel *pTChannel);
        ~SessionLogger(void);
        bool CloseFile(void);
        bool NotifyEnteringState(const gm::string& StateName);
        bool LogItem(const gm::string& ItemName,
                const gm::string& ItemValue = gm::string::Nullref());
        bool LogItem(const gm::string& ItemName, unsigned short ItemValue);
        bool LogItem(const gm::string& ItemName, float ItemValue);
        bool NotifyLeaveState(void);
        bool NotifyPropertySet(Property& prop);
        inline const gm::string& GetLogPath(void);
};
inline const gm::string&
SessionLogger::GetLogPath(void)
{
        return cLogPath;
}
#endif
```

```
/*
 * SessionMgr.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.25 $
 * $Date: 2000/12/07 03:20:11 $
 */
#ifndef SESSIONMGR_H
#define SESSIONMGR_H
#include "VPPUtils/CriticalSection.h"
#include "VPPUtils/Log.h"
#include "VPPUtils/Monitor.h"
#include "VPPUtils/Queue.h"
#include "VPPMsg/PropertyList.h"
#include "VPPMsg/Address.h"
#include "SessionNode.h"
#include <list>
#pragma warning(disable:4786)
class TelephonyServiceProvider;
class MediaServiceProvider;
class SpeechServiceProvider;
class Channel;
class TelephonyChannel;
class MediaChannel;
class SpeechChannel;
class TTSChannel;
class MTSServer;
class CallToken;
class RequestToken;
class VUISessionImpl;
class MTSServerImpl;
class __declspec(dllexport) SessionMgr : public Monitor
{
private:
   template class __declspec(dllexport) std::list<SessionNode *>;
   template class __declspec(dllexport) ::Queue<SessionNode *>;
   std::list<SessionNode *> cSessions;
   Queue<SessionNode *> zombieList;
   int maxZombieSize;
   MTSServer *cMTSServer;
   MTSServerImpl *cMTSServerImpl;
   void NotifyPropertiesSet(SessionNode *pNode);
   void NotifyPropertySet(SessionNode *pNode, Property& prop);
   PropertyList cDefaultProperties;
   void moveToZombieList(SessionNode* node);
   static SessionMgr* cSessionMgr;
public:
   SessionMgr(MTSServer *pMTSServer, MTSServerImpl *pMTSServerImpl);
   ~SessionMgr(void);
   void Initialize(void);
   bool CreateTrombone(TelephonyChannel *pCh1, TelephonyChannel *pCh2, bool FullDuplex);
```

```
    bool ResetRouting(TelephonyChannel *pCh);
    bool BreakTrombone(TelephonyChannel *pCh1, TelephonyChannel *pCh2);
    TelephonyChannel *GetFirstTelephonyChannel(void);
    VUISessionImpl *CreateSession(TelephonyChannel *pTCh, Address& Target, Address& CallingNumber);
    bool DestroySession(TelephonyChannel *pTCh);
    bool SetDefaultProperty(const gm::string& name, const gm::string& value);
    bool SetDefaultProperty(const gm::string& name, long value);
    bool GetDefaultProperty(const gm::string& name, gm::string& value);
    bool GetDefaultProperty(const gm::string& name, long& value);
    inline PropertyList& GetDefaultProperties(void);
    SessionNode *FindSessionNode(Channel *pCh);
    SessionNode *FindSessionNode(TelephonyChannel *pCh);
    SessionNode *FindSessionNode(MediaChannel *pCh);
    SessionNode *FindSessionNode(SpeechChannel *pCh);
    SessionNode *FindSessionNode(TTSChannel *pCh);
    SessionNode *FindSessionNode(CallToken& callToken);
    SessionNode *FindSessionNode(RequestToken& requestToken);
    inline MTSServer *GetMTSServer(void);
    inline MTSServerImpl *GetMTSServerImpl(void) {
            return cMTSServerImpl;
    }
    static void SetSessionMgr(SessionMgr* s);
    static SessionMgr* GetSessionMgr();
};
inline PropertyList&
SessionMgr::GetDefaultProperties(void)
{
        return cDefaultProperties;
}
inline MTSServer *
SessionMgr::GetMTSServer(void)
{
        return cMTSServer;
}
#endif
```

```
/*
 * SessionNode.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.32 $
 * $Date: 2000/12/07 03:20:11 $
 */
#ifndef SESSIONNODE_H
#define SESSIONNODE_H
#include "VPPMsg/PropertyList.h"
#include "VPPMsg/CDRInfo.h"
#include "VPPMsg/CDRInfoNtfy.h"
#include "VPPMsg/RecognitionDoneNtfy.h"
#include "VppMsg/RecordingDoneNtfy.h"
#include "VPPMsg/Address.h"
#include "SessionLogger.h"
#include "VPPUtils/CriticalSection.h"
#include "ConnectionListener.h"
#include "VPPUtils/VPPDebug.h"
#if 0
#include "VPPMsg/DynamicGrammarResultNtfy.h"
#endif
#include "VPPUtils/Monitor.h"
#include "VPPUtils/Synchronized.h"
#include "VPPUtils/Log.h"
#include "VPPUtils/IDLog.h"
#pragma warning(disable:4355)
class SetPropertyMessage;
class GetPropertyMessage;
class RegisterVUIBrokerMessage;
class WrappedMessage;
class PlaceCallMessage;
class DropCallMessage;
class ConnectCallsMessage;
class DisconnectCallsMessage;
class PlaceCallOnHoldMessage;
class AcceptCallMessage;
class RejectCallMessage;
class CancelRequestMessage;
class PlayMessage;
class StopPlayMessage;
class RecordMessage;
class StopRecordMessage;
class RemoveRecordingMessage;
class HandOffRecordingMessage;
class TTSMessage;
class CreateCompositePromptMessage;
class AppendCompositePromptPromptMessage;
class AppendCompositePromptSilenceMessage;
class AppendCompositePromptDTMFMessage;
class AppendCompositePromptTTSMessage;
```

```
class EndCompositePromptMessage;
class PlayCompositePromptMessage;
class StartRecognitionMessage;
class StopRecognitionMessage;
class RecognizeStringMessage;
class LogEnterStateMessage;
class LogItemMessage;
class LogLeaveStateMessage;
class CompositeMessage;
#if 0
class CreateDynamicGrammarMessage;
class DeleteDynamicGrammarMessage;
class AddPhraseToDynamicGrammarMessage;
class RemovePhraseFromDynamicGrammarMessage;
class QueryDynamicGrammarContentsMessage;
class InsertDynamicGrammarMessage;
#endif
class RecognitionDoneNotification;
class MediaServiceProvider;
class TelephonyServiceProvider;
class RecognitionResults;
class TelephonyChannel;
class MediaChannel;
class SpeechChannel;
class TTSChannel;
class SessionMgr;
class __declspec(dllexport) SessionNode : public Monitor, public IDLog
{
/*
        unsigned long cMutex;
        unsigned long cGoingAwayEvt;
        CriticalSection cCriticalSection;
        template class __declspec(dllexport) std::list<SessionLock *>;
        std::list<SessionLock *> cLocks;
*/
protected:
  bool cGoingAway;
  bool cDoWholeCallRecording;
  SessionMgr *cSessionMgr;
  // TelephonyServiceProvider *cTSP;
  void NotifyPropertiesSet(void);
  void NotifyPropertySet(Property& prop);
  TelephonyServiceProvider* GetTSP();
  MediaServiceProvider* GetMSP();
public:
  SessionNode(SessionMgr *pSessionMgr, TelephonyChannel *pTChannel);
  virtual ~SessionNode(void);
  unsigned long GetID();
/*
        void GetLock(SessionLock *pSessionLock, unsigned long OtherEvent);
        void ReleaseLock(SessionLock *pSessionLock);
*/
  bool SetProperties(PropertyList *pProperties);
  bool SetProperty(Property& prop);
  bool GetProperty(const gm::string& name, gm::string& value);
  bool GetProperty(const gm::string& name, long& value);
```

```
  bool UpdateCallToken(void);
  inline bool IsGoingAway(void);
  inline void SetGoingAway(bool bGoingAway);
  inline bool DoWholeCallRecording(void);
  bool HandleCancelRequest(CancelRequestMessage *pMsg, void *userdata);
  bool HandleDropCall(DropCallMessage *pMsg, void *userdata);
  bool HandlePlaceCallOnHold(PlaceCallOnHoldMessage *pMsg, void *userdata);
  bool HandleAcceptCall(AcceptCallMessage *pMsg, void *userdata);
  bool HandleRejectCall(RejectCallMessage *pMsg, void *userdata);
  bool HandleNotifyCDRInfo(CDRInfoNotification *pMsg, void *userdata);
  //
  // Media methods
  //
  bool HandlePlay(PlayMessage *pMsg, void *);
  bool HandleStopPlay(StopPlayMessage *pMsg, void *);
  bool HandleRecord(RecordMessage *pMsg, void *);
  bool HandleStopRecord(StopRecordMessage *pMsg, void *);
  bool HandleRemoveRecording(RemoveRecordingMessage *pMsg, void *);
  bool HandleTTS(TTSMessage *pMsg, void *);
  bool HandleCreateCompositePrompt(CreateCompositePromptMessage *pMsg,
                    void *);
  bool HandleAppendCompositePromptPrompt(AppendCompositePromptPromptMessage *pMsg, void *);
  bool HandleAppendCompositePromptSilence(AppendCompositePromptSilenceMessage *pMsg, void *);
  bool HandleAppendCompositePromptDTMF(AppendCompositePromptDTMFMessage *pMsg, void *);
  bool HandleAppendCompositePromptTTS(AppendCompositePromptTTSMessage *pMsg, void *);
  bool HandleEndCompositePrompt(EndCompositePromptMessage *pMsg, void *);
  bool HandlePlayCompositePrompt(PlayCompositePromptMessage *pMsg, void *);

  bool HandOffRecording(HandOffRecordingMessage *pMsg, void *userdata);
      // Convenience methods- methods that derived classes use to send
      // notifications back to clients
      //
  bool HandleReleaseOwnershipOfRecording(CallToken& ct, RECORDINGID RecordingID);
  bool GetRecordingFilename(CallToken& ct,
    RECORDINGID recordingID, gm::string& filename);
  //
  // Speech methods
  //
#if 0
  bool HandleStartRecognition(StartRecognitionMessage *pMsg, void *);
  bool HandleStopRecognition(StopRecognitionMessage *pMsg, void *);
  bool HandleRecognizeString(RecognizeStringMessage *pMsg, void *);
#endif
#if 0
  // Dynamic grammar methods
  bool HandleCreateDynamicGrammar(CreateDynamicGrammarMessage *pMsg, void *);
  bool HandleDeleteDynamicGrammar(DeleteDynamicGrammarMessage *pMsg, void *);
  bool HandleAddPhraseToDynamicGrammar(AddPhraseToDynamicGrammarMessage *pMsg, void *);
  bool HandleRemovePhraseFromDynamicGrammar(RemovePhraseFromDynamicGrammarMessage *pMsg, void *);
  bool HandleQueryDynamicGrammarContents(QueryDynamicGrammarContentsMessage *pMsg, void *);
  bool HandleInsertDynamicGrammar(InsertDynamicGrammarMessage *pMsg, void *);
#endif
  //
  // Logging methods
  //
  bool HandleLogEnterState(LogEnterStateMessage *pMsg, void *);
```

```
  bool HandleLogItem(LogItemMessage *pMsg, void *);
  bool HandleLogLeaveState(LogLeaveStateMessage *pMsg, void *);
  /************************************************
   *
   * Notification Methods
   *
   * ***********************************************/
  CMSMessage::SerialNumberT NotifyRequestCancelled(RequestToken& request);
  virtual CMSMessage::SerialNumberT NotifyCallProgress(RequestToken& request,
                                    CallProgressNotification::CallProgressT notification) = 0;
  virtual CMSMessage::SerialNumberT NotifyCallConnected(RequestToken& request, CallToken& callToken) = 0;
  virtual CMSMessage::SerialNumberT NotifyCallDropped(CallToken& ct) = 0;
  CMSMessage::SerialNumberT NotifySITDetected(CallToken& ct);
  CMSMessage::SerialNumberT NotifyCNGDetected(CallToken& ct);
  CMSMessage::SerialNumberT NotifyCallAccepted(CallToken& ct);
  CMSMessage::SerialNumberT NotifyCallRejected(CallToken& ct);
  virtual void NotifyIncomingCallRevoked(CallToken& ct) = 0;
  //
  // Notify Methods from Media
  //
  CMSMessage::SerialNumberT NotifyStoppedPlaying(CallToken& callToken);
  virtual CMSMessage::SerialNumberT NotifyRecording(CallToken& callToken, RECORDINGID recordingID, const
gm::string& filename) = 0;
  virtual CMSMessage::SerialNumberT NotifyRecordingDone(CallToken& callToken,
RecordingDoneNotification::RecordingDoneReason reason, unsigned long duration) = 0;
  CMSMessage::SerialNumberT NotifyStoppedRecording(CallToken& callToken);
  virtual CMSMessage::SerialNumberT NotifyCompositePromptSegmentPlayDone(CallToken& callToken) = 0;
  virtual CMSMessage::SerialNumberT NotifyCompositePromptPlayDone(CallToken& callToken) = 0;
  virtual CMSMessage::SerialNumberT NotifyDTMF(CallToken& callToken,
                              char dtmf, unsigned long timestamp) = 0;
  //
  // Notify Methods from SSP
  //
  virtual CMSMessage::SerialNumberT NotifyRecognitionDone(CallToken& ct,
                                    RecognitionResults *pResults,
                                    RecognitionDoneNotification::RecognitionStatus status) = 0;
#if 0
  CMSMessage::SerialNumberT NotifyBeginSpeech(CallToken& ct);
  CMSMessage::SerialNumberT NotifyEndSpeech(CallToken& ct);
#endif
  CMSMessage::SerialNumberT NotifyRecognitionStarted(CallToken& ct);
#if 0
  virtual CMSMessage::SerialNumberT NotifyDynamicGrammarCreated(CallToken& ct,
                                      DynamicGrammarResultNotification::Result result, const gm::string&
db_key) = 0;
  CMSMessage::SerialNumberT NotifyDynamicGrammarDeleted(CallToken& ct,
                                    DynamicGrammarResultNotification::Result result, const gm::string& db_key);
  CMSMessage::SerialNumberT NotifyPhraseAddedToDynamicGrammar(CallToken& ct,
                                      DynamicGrammarResultNotification::Result result, const gm::string& db_key,
                                      const gm::string& phrase_id);
  CMSMessage::SerialNumberT NotifyPhraseRemovedFromDynamicGrammar(CallToken& ct,
                                       DynamicGrammarResultNotification::Result result, const gm::string&
db_key,
                                       const gm::string& phrase_id);
  virtual CMSMessage::SerialNumberT NotifyDynamicGrammarContents(CallToken& ct,
```

```
                                        DynamicGrammarResultNotification::Result result, const gm::string&
db_key,
                                        const gm::string& phrase_id, const gm::string& phrase_contents) = 0;
  virtual CMSMessage::SerialNumberT NotifyDynamicGrammarInserted(CallToken& ct,
                                        DynamicGrammarResultNotification::Result result, const gm::string&
db_key,
                                        const gm::string& grammar_name) = 0;
#endif
  // Utility
  virtual CMSMessage::SerialNumberT NotifyMSPFailure(CallToken& ct, CMSMessage::SerialNumberT
FailureSerialNo, unsigned long Info) = 0;
  virtual CMSMessage::SerialNumberT NotifySSPFailure(CallToken& ct, CMSMessage::SerialNumberT
FailureSerialNo, unsigned long Info) = 0;
  virtual CMSMessage::SerialNumberT NotifyTSPFailure(CallToken& ct, CMSMessage::SerialNumberT
FailureSerialNo, unsigned long Info) = 0;
  virtual CMSMessage::SerialNumberT NotifyTSPFailure(CMSMessage::SerialNumberT FailureSerialNo, unsigned
long Info) = 0;
  //
  // Dropping Call
  //
  void NotifyAboutToDropCall(CallToken& ct);
  /*
   * Logger
   */
  bool HandleLogEnterState(const gm::string& StateName);
  bool HandleLogItem(const gm::string& ItemName,
                const gm::string& ItemValue);
  bool HandleLogLeaveState();
  /* public data member */
  PropertyList cProperties;
  SpeechChannel *cSpeechChannel;
  TelephonyChannel *cTelephonyChannel;
  MediaChannel *cPlayMediaChannel;
  MediaChannel *cRecordMediaChannel;
  MediaChannel *cOutboundRecordMediaChannel;
  MediaChannel *cInboundRecordMediaChannel;
  Address cTarget;                      // Destination number
  Address cCallingNumber;               // Origination number
  SessionLogger cSessionLogger;
  CDRInfo cCDRInfo;
  TTSChannel *cTTSChannel;
  unsigned long cID;
};
inline bool
SessionNode::IsGoingAway(void)
{
  return cGoingAway;
}
inline void
SessionNode::SetGoingAway(bool bGoingAway)
{
  Synchronized sync(this);
  cGoingAway = bGoingAway;
  if (bGoingAway) {
    sync.notifyAll();
    //::SetEvent((HANDLE)cGoingAwayEvt);
```

```
    }
}
inline bool
SessionNode::DoWholeCallRecording(void)
{
    return cDoWholeCallRecording;
}
#endif
```

```
/*
 * SetPropertyMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:21:31 $
 */
#ifndef SETPROPERTYMSG_H
#define SETPROPERTYMSG_H
#include "PropertyMsg.h"
#include "VPPDefs.h"
class __declspec(dllexport) SetPropertyMessage: public PropertyMessage
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(SetPropertyMessage, PropertyMessage);
        inline SetPropertyMessage(const gm::string& name, const gm::string& value);
#if 0
        virtual VirtualMachineID& GetMTSVirtualMachineID(void);
#endif
};
inline
SetPropertyMessage::SetPropertyMessage(const gm::string& name, const gm::string& value)
        : PropertyMessage(M_CMD_SETPROPERTY, name, value)
{
}
#endif
```

```
/*
 * SilenceSegment.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.6 $
 * $Date: 2000/12/07 03:20:12 $
 */
#ifndef SILENCESEGMENT_H
#define SILENCESEGMENT_H
#include "PromptSegment.h"
class SilenceSegment : public PromptSegment
{
        unsigned long cDuration;
public:
        inline SilenceSegment(unsigned long duration);
        virtual ~SilenceSegment(void);
        virtual bool Play(MediaChannel *pMediaChannel,
                CallToken& ct);
        inline void Append(unsigned long duration);
};

inline
SilenceSegment::SilenceSegment(unsigned long duration)
        : PromptSegment(PromptSegment::ST_Silence)
{
        cDuration = duration;
}
inline void
SilenceSegment::Append(unsigned long duration)
{
        cDuration += duration;
}
#endif
```

```
/*
 * SITDetectedNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:31 $
 */
#ifndef SITDETECTEDNTFY_H
#define SITDETECTEDNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) SITDetectedNotification : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(SITDetectedNotification,
                OneCallTokenMessageBase);
        inline SITDetectedNotification(CallToken& token);
};
inline
SITDetectedNotification::SITDetectedNotification(CallToken& token)
        : OneCallTokenMessageBase(M_NTFY_SITDETECTED, token)
{
}
#endif
```

```
/*
 * Slot.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.6 $
 * $Date: 2000/12/07 03:21:32 $
 */
#ifndef SLOT_H
#define SLOT_H
#include "VPPUtils/GMString"
#include <list>
#pragma warning(disable:4786)
class VPPMessage;
class __declspec(dllexport) Slot
{
public:
        typedef enum
        {
                T_String,
                T_Int,
                T_Structure
        } Type;
private:
        gm::string cName;
        Type cType;
        float cConfidence;
        gm::string cStringValue;
        int cIntValue;
        template class __declspec(dllexport) std::list<Slot *>;
        std::list<Slot *> cStructureValue;
public:
        inline Slot(const gm::string& name, const gm::string& value, float confidence = 0);
        inline Slot(const gm::string& name, int value, float confidence = 0);
        inline Slot(const gm::string& name, float confidence = 0);
        inline Slot(void);
        virtual ~Slot(void);
        inline void SetStringValue(const gm::string& value);
        inline void SetIntValue(int value);
        void AddStructSlotValue(Slot *pSlot);
        inline const gm::string& GetName(void);
        inline Type GetType(void);
        float GetConfidence(void);
        inline const gm::string& GetString(void);
        inline int GetInt(void);
        Slot *GetStructSlot(int index);
        void Serialize(VPPMessage *);
        void Deserialize(VPPMessage *);
};
inline void
Slot::SetStringValue(const gm::string& value)
{
```

```
        cType = T_String;
        cStringValue = value.c_str();
}
inline void
Slot::SetIntValue(int value)
{
        cType = T_Int;
        cIntValue = value;
}
inline
Slot::Slot(const gm::string& name, const gm::string& value, float confidence)
{
        cName = name.c_str();
        SetStringValue(value);
        cConfidence = confidence;
}
inline
Slot::Slot(const gm::string& name, int value, float confidence)
{
        cName = name.c_str();
        SetIntValue(value);
        cConfidence = confidence;
}
inline
Slot::Slot(const gm::string& name, float confidence)
{
        cName = name.c_str();
        cType = T_String;
        cConfidence = confidence;
}
inline
Slot::Slot(void)
{
        cType = T_String;
        cConfidence = 0.0;
}
inline const gm::string&
Slot::GetName(void)
{
        return cName;
}
inline Slot::Type
Slot::GetType(void)
{
        return cType;
}
inline float
Slot::GetConfidence(void)
{
        return cConfidence;
}
inline const gm::string&
Slot::GetString(void)
{
        return cStringValue;
}
```

```
inline int
Slot::GetInt(void)
{
        return cIntValue;
}
#endif
```

```
/*
 * SpeechChannel.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.30 $
 * $Date: 2000/12/07 03:20:12 $
 */
#ifndef SPEECHCHANNEL_H
#define SPEECHCHANNEL_H
#include "Channel.h"
#include "VPPUtils/CriticalSection.h"
#include "VPPUtils/GMString"
#include "VPPMsg/RecognitionDoneNtfy.h"
#include "SampleListener.h"
#if 0
#include "VPPMsg/DynamicGrammarResultNtfy.h"
#endif
#include <VPPUtils/Timer.h>
#include <MTSLib/GrammarHandle.h>
class MediaChannel;
class StartRecognitionMessage;
class StopRecognitionMessage;
class RecognitionResults;
class SpeechServiceProvider;
class RecognizeStringMessage;
class DynamicGrammar;
#if 0
class CreateDynamicGrammarMessage;
class DeleteDynamicGrammarMessage;
class AddPhraseToDynamicGrammarMessage;
class RemovePhraseFromDynamicGrammarMessage;
class QueryDynamicGrammarContentsMessage;
class InsertDynamicGrammarMessage;
#endif
class Slot;
class __declspec(dllexport) SpeechChannel : public Channel, public SampleListener
{
private:
        CMSMessage::SerialNumberT cMsgSerialNumber; // used currently only for recognition NotifyFailure
public:
        typedef enum
        {
                S_Idle,
                S_RecognizingPreSpeech,
                S_RecognizingSpeech,
                S_PostEndOfSpeech,
                S_RecognizingDTMF,
                S_RecognizingPostDTMF,
                S_RecognizingString,
        } ChannelState;
protected:
```

```
        CriticalSection cCriticalSection;
        ChannelState cState;
    Timer::TimerID BOS_TimerID;
    Timer::TimerID TMS_TimerID;
    Timer::TimerID DTMF_TimerID;
        gm::string cDTMFString;
        unsigned short cDTMFMaxDigits;
        unsigned short cDTMFDigits;
        GrammarHandle cLastGrammar;
        bool cBargeInAllowed;
        bool cMagicWord;
        unsigned short cRound;
        gm::string cRecordingFilename;
    void BOS_Timeout();
    void TMS_Timeout();
    void DTMF_Timeout();

        //
        // Pure-virtual methods that must be implemented by the subclass.  All
        // return one of the SSPF_xxx codes from vppmsg.h
        //
#if 0
        virtual unsigned long StartRecognition(const gm::string& grammar) = 0;
        virtual unsigned long RecognizeString(const gm::string& grammar,
                        const gm::string& str) = 0;
#else
        virtual unsigned long StartRecognition(const GrammarHandle& grammar) = 0;
        virtual unsigned long RecognizeString(const GrammarHandle& grammar,
                        const gm::string& str) = 0;
#endif
        virtual unsigned long PushSamples(const unsigned char *pSamples, unsigned int numbytes) = 0;
        virtual unsigned long StopRecognition(void) = 0;
        virtual unsigned long AbortRecognition(void) = 0;
        virtual bool StartLogging(void) = 0;
        virtual bool LogUtteranceFilename(void) = 0;
        virtual bool StopLogging(void) = 0;
#if 0
        virtual unsigned long CreateDynamicGrammar(const gm::string& db_key, const gm::string& gsl) = 0;
        virtual unsigned long DeleteDynamicGrammar(const gm::string& db_key) = 0;
        virtual unsigned long AddPhraseToDynamicGrammar(const gm::string& db_key,
                const gm::string& phrase_id, const gm::string& phrase_text,
                const gm::string& phrase_nl, const unsigned long probability) = 0;
        virtual unsigned long RemovePhraseFromDynamicGrammar(const gm::string& db_key,
                const gm::string& phrase_id) = 0;
        virtual unsigned long QueryDynamicGrammarContents(const gm::string& db_key,
                const gm::string& phrase_id) = 0;
        virtual unsigned long InsertDynamicGrammar(const gm::string& db_key,
                const gm::string& grammar_name) = 0;
        unsigned long DoStartRecognition(const gm::string& grammar);
#endif
        unsigned long DoStartRecognition(const GrammarHandle& handle);
        void DumpSlot(Slot *pSlot, gm::string& str);
        //
        // Notification methods that derived classes call to notify the
        // base class of asynchronous events
        //
```

```
        void NotifyStartOfSpeech(void);
        void NotifyEndOfSpeech(void);
        void NotifyRecognitionDone(RecognitionResults *pResults,
                RecognitionDoneNotification::RecognitionStatus status);
  void NotifySettled(void);
private:
#ifdef ENABLE_PERF_MEASURE
        unsigned long cBOSTime;
  inline unsigned long GetBOSTime(void);
#endif
        inline SpeechServiceProvider *GetSSP(void);
        void SetState(ChannelState state);
        void KillTimers(void);
        void StopTimer(Timer::TimerID &id);
        void AppendDTMFChar(char which);
        void StartBOSTimer(void);
        void StartTMSTimer(void);
        void StartDTMFTimer(void);
        void EndDTMFRecognition(void);
public:
        SpeechChannel(ServiceProvider *pServiceProvider, const char *logType, int id, bool RequiresSettlement =
false);
        virtual ~SpeechChannel(void);
        ChannelState GetState(void);
        void NotifyPlayStarted(void);
        void NotifyPlayDone(void);
        void NotifyPlayStopped(void);
        void NotifyDTMF(char which);
        virtual void NotifySample(const unsigned char *pSample, unsigned int numBytes);
        virtual void NotifyEndOfData(int reason);
        virtual void NotifyPropertySet(Property& prop);
        bool NotifyStartLogging(void);
        bool NotifyStopLogging(void);
#if 0
        virtual void NotifyDynamicGrammarCreated(DynamicGrammarResultNotification::Result result,
                const gm::string& db_key);
        virtual void NotifyDynamicGrammarDeleted(DynamicGrammarResultNotification::Result result,
                const gm::string& db_key);
        virtual void NotifyPhraseAddedToDynamicGrammar(DynamicGrammarResultNotification::Result result,
                const gm::string& db_key, const gm::string& phrase_id);
        virtual void NotifyPhraseRemovedFromDynamicGrammar(DynamicGrammarResultNotification::Result
result,
                const gm::string& db_key, const gm::string& phrase_id);
        virtual void NotifyDynamicGrammarContents(DynamicGrammarResultNotification::Result result,
                const gm::string& db_key, const gm::string& phrase_id, const gm::string& phrase_contents);
        virtual void NotifyDynamicGrammarInserted(DynamicGrammarResultNotification::Result result,
                const gm::string& db_key, const gm::string& grammar_name);
#endif
#ifdef ENABLE_PERF_MEASURE
        inline void SetBOSTime(unsigned long time);
        inline unsigned long GeBtOSTime(void);
#endif
// new Speech API
#if 0
        bool HandleStartRecognition(SessionNode *pSession, StartRecognitionMessage *pMsg, MediaChannel
*pPlayChannel, MediaChannel *pRecordChannel,
```

```
                void *userdata);
        bool HandleRecognizeString(SessionNode *pSession, RecognizeStringMessage *pMsg, void *userdata);
#else
        unsigned long HandleStartRecognition(SessionNode *pSession,
                                const GrammarHandle& gram,
                                MediaChannel *pPlayChannel,
                                MediaChannel *pRecordChannel);
        virtual unsigned long GetGrammarHandle(DynamicGrammar& gram,
                                 GrammarHandle& handle) = 0;
        unsigned long HandleRecognizeString(SessionNode *pSession,
                               const GrammarHandle& gram,
                               const gm::string& str);
#endif
        bool HandleStopRecognition(SessionNode *pSession, StopRecognitionMessage *pMsg, void *userdata);
#if 0
        bool HandleCreateDynamicGrammar(SessionNode *pSession, CreateDynamicGrammarMessage *pMsg, void
*userdata);
        bool HandleDeleteDynamicGrammar(SessionNode *pSession, DeleteDynamicGrammarMessage *pMsg, void
*userdata);
        bool HandleAddPhraseToDynamicGrammar(SessionNode *pSession,
AddPhraseToDynamicGrammarMessage *pMsg, void *userdata);
        bool HandleRemovePhraseFromDynamicGrammar(SessionNode *pSession,
RemovePhraseFromDynamicGrammarMessage *pMsg, void *userdata);
        bool HandleQueryDynamicGrammarContents(SessionNode *pSession,
QueryDynamicGrammarContentsMessage *pMsg, void *userdata);
        bool HandleInsertDynamicGrammar(SessionNode *pSession, InsertDynamicGrammarMessage *pMsg, void
*userdata);
#endif
        virtual bool NotifyCallAboutToBeDropped(SessionNode* pSession, CallToken& callToken);
};
inline SpeechServiceProvider *
SpeechChannel::GetSSP(void)
{
        return (SpeechServiceProvider *)cServiceProvider;
}
#ifdef ENABLE_PERF_MEASURE
inline void
SpeechChannel::SetBOSTime(unsigned long time)
{
        cBOSTime = time;
}
inline unsigned long
SpeechChannel::GetBOSTime(void)
{
        return cBOSTime;
}
#endif
#endif
```

```
/*
 * SpeechServiceProvider.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.18 $
 * $Date: 2000/12/07 03:20:13 $
 */
#ifndef SPEECHSERVICEPROVIDER_H
#define SPEECHSERVICEPROVIDER_H
#include "ServiceProvider.h"
#include "VPPMsg/DynamicGrammarResultNtfy.h"
#include "VPPUtils/Timer.h"
class SpeechChannel;
class StartRecognitionMessage;
class StopRecognitionMessage;
class RecognitionResults;
class RecognizeStringMessage;
class RecognitionDoneNotification;
class CreateDynamicGrammarMessage;
class DeleteDynamicGrammarMessage;
class AddPhraseToDynamicGrammarMessage;
class RemovePhraseFromDynamicGrammarMessage;
class QueryDynamicGrammarContentsMessage;
class InsertDynamicGrammarMessage;
class __declspec(dllexport) SpeechServiceProvider : public ServiceProvider
{
protected:
   Timer timer;
public:
        SpeechServiceProvider(MTSServer *pServer, const char *logType);
        SpeechChannel *FindAvailableChannel(void);
        virtual ~SpeechServiceProvider(void);
    inline Timer& GetTimer() {
      return timer;
    }
        // Handlers - methods that handle requests and notifications from
        // clients
        //
};
typedef SpeechServiceProvider *(*CREATESSPFUNC)(MTSServer *);
extern "C" __declspec(dllexport)
SpeechServiceProvider *CreateSpeechServiceProvider(MTSServer *);
#endif
```

```
/*
 * StartRecognitionMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:21:32 $
 */
#ifndef STARTRECOGNITIONMSG_H
#define STARTRECOGNITIONMSG_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
#include "VPPUtils/GMString"
class __declspec(dllexport) StartRecognitionMessage :
        public OneCallTokenMessageBase
{
        gm::string cGrammar;
        bool cMagicWord;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(StartRecognitionMessage, OneCallTokenMessageBase);
        inline StartRecognitionMessage(CallToken& token, const gm::string grammar,
                bool MagicWord);
        inline const gm::string& GetGrammar(void);
        inline bool GetMagicWord(void);
};
inline
StartRecognitionMessage::StartRecognitionMessage(CallToken& token, const gm::string grammar,
        bool MagicWord): OneCallTokenMessageBase(M_CMD_STARTRECOGNITION, token)
{
        cGrammar = grammar.c_str();
        cMagicWord = MagicWord;
}
inline const gm::string&
StartRecognitionMessage::GetGrammar(void)
{
        return cGrammar;
}
inline bool
StartRecognitionMessage::GetMagicWord(void)
{
        return cMagicWord;
}
#endif
```

```
/*
 * StoppedPlayingNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:33 $
 */
#ifndef STOPPEDPLAYINGNTFY_H
#define STOPPEDPLAYINGNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) StoppedPlayingNotification : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(StoppedPlayingNotification, OneCallTokenMessageBase);
        inline StoppedPlayingNotification(CallToken& token);
};
inline
StoppedPlayingNotification::StoppedPlayingNotification(CallToken& token)
        : OneCallTokenMessageBase(M_NTFY_STOPPEDPLAYING, token)
{
}
#endif
```

```
/*
 * StoppedRecordingNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:34 $
 */
#ifndef STOPPEDRECORDINGNTFY_H
#define STOPPEDRECORDINGNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) StoppedRecordingNotification : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(StoppedRecordingNotification, OneCallTokenMessageBase);
        inline StoppedRecordingNotification(CallToken& token);
};
inline
StoppedRecordingNotification::StoppedRecordingNotification(CallToken& token)
        : OneCallTokenMessageBase(M_NTFY_STOPPEDRECORDING, token)
{
}
#endif
```

```
/*
 * StopPlayMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:33 $
 */
#ifndef STOPPLAYMSG_H
#define STOPPLAYMSG_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
#include "VPPUtils/GMString"
class __declspec(dllexport) StopPlayMessage : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(StopPlayMessage, OneCallTokenMessageBase);
        inline StopPlayMessage(CallToken& token);
};
inline
StopPlayMessage::StopPlayMessage(CallToken& token)
        : OneCallTokenMessageBase(M_CMD_STOPPLAY, token)
{
}
#endif
```

```
/*
 * StopRecognitionMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:33 $
 */
#ifndef STOPRECOGNITIONMSG_H
#define STOPRECOGNITIONMSG_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) StopRecognitionMessage : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(StopRecognitionMessage, OneCallTokenMessageBase);
        inline StopRecognitionMessage(CallToken& token);
};
inline
StopRecognitionMessage::StopRecognitionMessage(CallToken& token)
        : OneCallTokenMessageBase(M_CMD_STOPRECOGNITION, token)
{
}
#endif
```

```
/*
 * StopRecordMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:33 $
 */
#ifndef STOPRECORDMSG_H
#define STOPRECORDMSG_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
#include "VPPUtils/GMString"
class __declspec(dllexport) StopRecordMessage : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(StopRecordMessage, OneCallTokenMessageBase);
        inline StopRecordMessage(CallToken& token);
};
inline
StopRecordMessage::StopRecordMessage(CallToken& token)
        : OneCallTokenMessageBase(M_CMD_STOPRECORD, token)
{
}
#endif
```

```
/*
 * Synchronize.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:22:25 $
 */
#ifndef SYNCHRONIZE_H
#define SYNCHRONIZE_H
#include "CriticalSection.h"
class __declspec(dllexport) Synchronize
{
        CriticalSection& cCriticalSection;
public:
        inline Synchronize(CriticalSection& criticalsection);
        inline ~Synchronize(void);
};
inline
Synchronize::Synchronize(CriticalSection& criticalsection)
        : cCriticalSection(criticalsection)
{
        cCriticalSection.Enter();
}
inline
Synchronize::~Synchronize(void)
{
        cCriticalSection.Leave();
}
#endif
```

```
/*
 * Synchronized.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:22:25 $
 */
#ifndef _SYNCHRONIZED_H
#define _SYNCHRONIZED_H
#include "VPPUtils/Monitor.h"
class __declspec(dllexport) Synchronized
{
   Monitor& monitor;
   int nest;
public:
   inline Synchronized(Monitor& m) : monitor(m), nest(-1) {
      monitor.enter();
   }
   inline Synchronized(Monitor* m) : monitor(*m), nest(-1) {
      monitor.enter();
   }
   inline ~Synchronized(void) {
      if (nest != -1) {
         join();
      }
      monitor.leave();
   }
   inline void wait(long duration = 0) {
      monitor.wait(duration);
   }
   inline void yield() {
      nest = monitor.yield();
   }
   inline void join() {
      monitor.join(nest);
      nest = -1;
   }
   inline void notify() {
      monitor.notify();
   }
   inline void notifyAll() {
      monitor.notifyAll();
   }
};
#endif
```

```
/*
 * t2sspmsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:37 $
 */
#ifndef T2SSPMSG_H
#define T2SSPMSG_H
#include "VPPMsg/OneCallTokenMsgBase.h"
#include "VPPMsg/VPPDefs.h"
#include "VPPUtils/GMString"
#include "VPPUtils/GMLangID.h"
#include "CMS/CMSMsg.h"
class ServiceProvider;
//|-----------------------------------------|
//| this is the a server T2S request        |
//|-----------------------------------------|
class __declspec(dllexport) T2SSPMessage: public VPPMessage
{
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        gm::string cText;
        unsigned char initialize;
        DECLARE_CMSMESSAGE_SUBCLASS(T2SSPMessage, VPPMessage);
        T2SSPMessage( const gm::string& text,unsigned char init);
        inline gm::string& GetText(void);
};
inline gm::string&
T2SSPMessage::GetText(void)
{
        return cText;
}
class __declspec(dllexport) T2SSPstartMessage: public VPPMessage
{
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        unsigned char channel;
        DECLARE_CMSMESSAGE_SUBCLASS(T2SSPstartMessage, VPPMessage);
        T2SSPstartMessage(void);
};
// This  a STOP T2S message
class __declspec(dllexport) T2SSPstopMessage: public VPPMessage
{
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
```

```
public:

        DECLARE_CMSMESSAGE_SUBCLASS(T2SSPstopMessage, VPPMessage);
        T2SSPstopMessage(void);
};
//|-----------------------------------------|
//| this is the reply from the server.... T2S |
//|-----------------------------------------|
class __declspec(dllexport) T2SSPMessageReply: public VPPMessage
{


protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        unsigned long     csynSize;
        char      cdataDone;
        char      *csynth_buffer;
        DECLARE_CMSMESSAGE_SUBCLASS(T2SSPMessageReply, VPPMessage);
        inline T2SSPMessageReply(unsigned long synSize, char *synth_buffer, bool dataDone);
        inline char *Getsynth(void);
        inline unsigned long& GetsynSize(void);
        inline char&  GetdataDone(void);
        T2SSPMessageReply::~T2SSPMessageReply(void);
};
inline
T2SSPMessageReply::T2SSPMessageReply(unsigned long synSize ,char *synth_buffer, bool dataDone)
{
        csynth_buffer     = synth_buffer;
        csynSize          = synSize;
        cdataDone                 = (char )dataDone;
}
inline
char *
T2SSPMessageReply::Getsynth(void)
{
        return csynth_buffer;
}
inline
unsigned long&
T2SSPMessageReply::GetsynSize(void)
{
        return csynSize;
}
inline
char&
T2SSPMessageReply::GetdataDone(void)
{
        return cdataDone;
}
#endif
```

```
/*
 * TelephonyChannel.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.36 $
 * $Date: 2000/12/07 03:20:14 $
 */
#ifndef TELEPHONYCHANNEL_H
#define TELEPHONYCHANNEL_H
#include "SCChannel.h"
#include "VPPMsg/CallProgressNtfy.h"
#include "VPPMsg/RequestToken.h"
#include "VPPUtils/Timer.h"
#include "VPPUtils/Monitor.h"
class TelephonyServiceProvider;
class PlaceCallMessage;
class DropCallMessage;
class ConnectCallsMessage;
class PlaceCallOnHoldMessage;
class AcceptCallMessage;
class RejectCallMessage;
class CancelRequestMessage;
class DisconnectCallsMessage;
class Address;
class ProcessShutdownCompleteNotification;
class __declspec(dllexport) TelephonyChannel : public SCChannel, public Monitor
{
public:
        typedef enum
        {
                S_Disconnected,
                S_IncomingNotified,
                S_OutgoingDialed,
                S_Connected
        } ChannelState;
        typedef enum
        {
                S_InService,
                S_OutOfService,
                S_Maintenance
        } ChannelStatus;
        enum
  {
                C_Incoming = 0x1,
                C_Outgoing = 0x2,
    C_Unused  = 0x4
        };
        typedef enum {
                RC_UnassignedNumber,  // invalid called party number
                RC_UserBusy,                    // called party is busy
                RC_NetworkCongestion,  // cannot establish connection due to volume of traffic on network
```

```
                RC_SendSIT                          // send a Special Information Tone
        } RejectCause;
private:
        Timer::TimerID timerID;
    void StopTimer();
    void InCall_Timeout();
    void OutCall_Timeout();
        inline TelephonyServiceProvider *GetTSP(void);
        void SetState(ChannelState state);
  void WaitForSessionNode(void);
protected:
        ChannelState cState;
        ChannelStatus cStatus;
        unsigned short cCaps;
        RequestToken cCurrentRequestToken;
        //
        // Pure-virtual methods that must be implemented by the subclass.  All
        // return one of the TSPF_xxx codes from vppmsg.h
        //
        virtual unsigned long PlaceCall(const gm::string& calledNumber,
                const gm::string& callingNumber, unsigned char *uui = NULL, int uuiLength = 0) = 0;
        virtual unsigned long CancelRequest(void) = 0;
        virtual unsigned long DropCall(void) = 0;
        virtual unsigned long PlaceCallOnHold(void) = 0;
        virtual unsigned long AcceptCall(void) = 0;
        virtual unsigned long RejectCall(RejectCause cause) = 0;
        //
        // Notification methods that derived classes call to notify the
        // base class of asynchronous events
        //
        void NotifyCallConnected(void);
        void NotifySITDetected(void);
        void NotifyCNGDetected(void);
        void NotifyIncomingCall(Address& DNIS, Address& ANI, char *UUI, int length);
        void NotifyAboutToDropCall(void);
        void NotifyCallDropped(void);
  void NotifySettled(void);
        // Other protected methods
        void InitializeCurrentCallToken(void);
        void ResetChannel(void);
protected:
  // by M.O
  friend class TelephonyServiceProvider;
        void NotifyCallProgress(CallProgressNotification::CallProgressT ntfy);
public:
        TelephonyChannel(ServiceProvider *pServiceProvider, const char *logType, int id, bool RequiresSettlement
= false);
        virtual ~TelephonyChannel(void);
    // *************************
    // NEW API
        bool HandleCancelRequest(SessionNode* pSession,
                CancelRequestMessage *pMsg, void *userdata);
        bool HandleDropCall(SessionNode* pSession, DropCallMessage *pMsg, void *userdata);
        bool HandlePlaceCallOnHold(SessionNode* pSession, PlaceCallOnHoldMessage *pMsg, void *userdata);

        bool HandleAcceptCall(SessionNode* pSession,
```

```
                AcceptCallMessage *pMsg, void *userdata);

        bool HandleRejectCall(SessionNode* pSession, RejectCallMessage *pMsg, void *userdata);

        bool HandleNotifySuspended(SessionNode* pSession,
                VPPMessage *pMsg, void *userdata);
    // END NEW API
    // ************************
    // ************************
    // NEW NEW API
        unsigned long HandlePlaceCall(const gm::string& calledNumber,
                const gm::string& callingNumber,
                                                unsigned char *uui, int uuiLength,
RequestToken& requestToken);
        unsigned long HandleCreateTrombone(ConnectCallsMessage *pMsg, void *userdata,
                        TelephonyChannel *pOther);

        unsigned long HandleBreakTrombone(DisconnectCallsMessage *pMsg, void *userdata);

        unsigned long HandlePlaceCallOnHold(PlaceCallOnHoldMessage *pMsg, void *userdata);
    // END OF NEW NEW API
    // ************************
        unsigned long DoCancelCall(void);
        unsigned long DoDropCall(void);
        virtual bool IsAvailable(void);
        inline ChannelState GetState(void);
        inline ChannelStatus GetStatus(void);
        inline void SetStatus(ChannelStatus status);
  virtual bool SetPhysicalChannelStatus(TelephonyChannel::ChannelStatus status) = 0;
        void SetStatusAndNotifyTSP(ChannelStatus status);
        inline unsigned short GetCapabilities(void) const;
        inline void SetCapabilities(unsigned short caps);
        inline RequestToken& GetCurrentRequestToken(void);
        void ResetRouting(void);
        virtual void NotifyPropertySet(Property& prop);
        void CleanupResources(void);
        virtual const char *GetIDString(void) = 0;
        SessionNode* getSessionNode() { if(Channel::GetState() == Channel::S_InSession) return cSession; else
return NULL; };
};
inline TelephonyChannel::ChannelState
TelephonyChannel::GetState(void)
{
        return cState;
}
inline TelephonyChannel::ChannelStatus
TelephonyChannel::GetStatus(void)
{
        return cStatus;
}
inline void
TelephonyChannel::SetStatus(ChannelStatus status)
{
        cStatus = status;
}
inline unsigned short
```

```
TelephonyChannel::GetCapabilities(void) const
{
        return cCaps;
}
inline RequestToken&
TelephonyChannel::GetCurrentRequestToken(void)
{
        return cCurrentRequestToken;
}
inline TelephonyServiceProvider *
TelephonyChannel::GetTSP(void)
{
        return (TelephonyServiceProvider *)cServiceProvider;
}
inline void
TelephonyChannel::SetCapabilities(unsigned short caps)
{
        cCaps = caps;
}
#endif
```

```
/*
 * TelephonyServiceProvider.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.33 $
 * $Date: 2000/12/07 03:20:14 $
 */
#ifndef TELEPHONYSERVICEPROVIDER_H
#define TELEPHONYSERVICEPROVIDER_H
#include "ServiceProvider.h"
#include "VPPMsg/CallProgressNtfy.h"
#include "TelephonyChannel.h"
#include "VPPUtils/Timer.h"
class OutboundCallRequest;
class TelephonyChannel;
class PlaceCallMessage;
class DropCallMessage;
class PlaceCallOnHoldMessage;
class AcceptCallMessage;
class RejectCallMessage;
class CancelRequestMessage;
class DisconnectCallsMessage;
class ConnectCallsMessage;
class Address;
class CDRInfo;
class ProcessShutdownCompleteNotification;
// class SessionLock;
class __declspec(dllexport) TelephonyServiceProvider : public ServiceProvider
{
private:
   CriticalSection cCriticalSection;
   Timer timer;
   std::list<Channel *>::iterator cIter;
protected:
        bool cIncomingRequiresMediaChannel;
        unsigned long cTotalNumOutboundChans;
public:
        TelephonyServiceProvider(MTSServer *pServer, const char *logType);
        virtual ~TelephonyServiceProvider(void);
        virtual bool PostSetup(gm::string registryParent);
        virtual bool Restart(void) = 0;
   virtual bool BeginService(void) = 0;
        TelephonyChannel *FindAvailableChannelForIncomingCall(void);
        TelephonyChannel *FindAvailableChannelForOutgoingCall(void);
   //TelephonyChannel * GrabAvailableChannelForOutgoingCall(void);
     //************************
     // NEW API
        virtual unsigned long HandlePlaceCall(const char* number,
                        const char* ani,
                                                unsigned char *uui,
                                                int uuiLength,
```

```
                        OutboundCallRequest* request,
                        CDRInfo& cdr);
        virtual unsigned long HandleConnectCalls(ConnectCallsMessage *pMsg, void *userdata);
        virtual unsigned long HandleDisconnectCalls(DisconnectCallsMessage *pMsg, void *userdata);
    // END NEW API
    //*************************
        // Convenience methods- methods that derived classes use to send
        // notifications back to clients
        //
        CMSMessage::SerialNumberT NotifyIncomingCall(CallToken& callToken,
                Address& DNIS, Address& ANI, char *UUI, int uuiLength);
        // void NotifyAboutToDropCall(ClientID& VUIClientID, CallToken& ct);
        inline bool IncomingRequiresMediaChannel(void);
        void CleanupAllResources(void);
        void NotifyChannelStatus(TelephonyChannel::ChannelStatus status, const TelephonyChannel& tChannel);
        bool GetOutboundChannelAvailable(void);
        void NotifyCallBooted(SessionNode *session);
        inline unsigned long GetTotalNumOutboundChans(void);
    inline Timer& GetTimer() {
      return timer;
    }
};
inline bool
TelephonyServiceProvider::IncomingRequiresMediaChannel(void)
{
        return cIncomingRequiresMediaChannel;
}
inline unsigned long
TelephonyServiceProvider::GetTotalNumOutboundChans(void)
{
        return cTotalNumOutboundChans;
}
typedef TelephonyServiceProvider *(*CREATETSPFUNC)(MTSServer *);
extern "C" __declspec(dllexport)
TelephonyServiceProvider *CreateTelephonyServiceProvider(MTSServer *);
#endif
```

```
/*
 * Thread.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:22:26 $
 */
#ifndef THREAD_H
#define THREAD_H
#include <windows.h>
class __declspec(dllexport) Thread {
public:
   Thread(int p = THREAD_PRIORITY_NORMAL, bool a=true) : priority(p), autodelete(a) {}
   virtual ~Thread(void) {}
   void setPriority(int priority);
   void start(void);
   void join();
   virtual void run(void) = 0;
   bool isAutodelete() {
      return autodelete;
   }
         void terminate( int exit_code =-1 ) { ::TerminateThread(t_handle, exit_code); }
         bool isRunning( DWORD wait =1 ) { return ::WaitForSingleObject(t_handle,wait) == WAIT_TIMEOUT; }
private:
   int priority;
   bool   autodelete;
protected:
   HANDLE    t_handle;
};
/**
 * Template based Thread, whcih will invoke a method on paticular
 * object.
 * <code>
 * class A {
 * public:
 *    void myFunction();
 * };
 *
 * void func() {
 *    A a;
 *    (new ThreadT<A>(&a, A::myFunction)).start();
 * }
 * </code>
 */
template<class T> class __declspec(dllexport) ThreadT : public Thread {
public:
   typedef void (T::*RUN)();
   ThreadT(T* t, RUN r, int prior = THREAD_PRIORITY_NORMAL, bool autodel=true )
         : Thread(prior, autodel), _this(t), _run(r) {}

    virtual ~ThreadT(void) {}
```

```
    virtual void run(void) {
      (_this->*_run)();
    }
private:
    T* _this;
    RUN _run;
};
/**
 * Java like thread library that uses Runnable interface.
 */
class __declspec(dllexport) Runnable {
public:
    virtual void run() = 0;
};
/**
 *
 */
class __declspec(dllexport) ThreadR : public Thread {
public:
    ThreadR(Runnable&r) : runnable(&r) {}
    virtual ~ThreadR(void) {}
    virtual void run(void);
private:
    Runnable* runnable;
};
#endif
```

```
/*
 * Timer.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.8 $
 * $Date: 2000/12/07 03:22:26 $
 */
#ifndef  _TIMER_H
#define _TIMER_H
#include <windows.h>
#include <stdio.h>
#include "Monitor.h"
class __declspec(dllexport) TimerEvent;
/**
 * Binary search version of Vector based Queue.
 * This performs quite well until the number of events gets
 * beyond 10000. We should use Heap based Priority queue
 * if we found we have to handle more than 10000 events.
 */
class __declspec(dllexport) Timer : public Monitor {
public:
   typedef unsigned long Timer::TimerID;
   class __declspec(dllexport) Filter {
   public:
      virtual bool accept(TimerEvent* ev) = 0;
      virtual bool stop() = 0;
   };
   Timer();
   virtual ~Timer();
   Timer::TimerID add(TimerEvent* event);
   bool remove(Timer::TimerID id);
   int remove(Filter &flt);
   void stop();
   void run();
   void start(int numberOfThread = 1,
         int priority = THREAD_PRIORITY_NORMAL);
private:
   typedef TimerEvent* TE_PTR;
   TimerEvent* dequeue();
   unsigned long valueAt(int x);
   int bsearch(TimerEvent* event);
   bool _running;
   int numOfThreads;
   // List Operations. this may be replaced with STL library instead later.
   void arraycopy(TE_PTR* src, int src_pos,
            TE_PTR* to, int to_pos, int length);
   void insertElementAt(TimerEvent* ev, int at);
   void removeElementAt(int at);
   TimerEvent* lastElement();
   TE_PTR* elementData;
   int bsize;
```

```
  int elementCount;
};
class __declspec(dllexport) TimerEvent {
protected:
  TimerEvent(unsigned long w) : when(w), id(0), name(NULL) {};
  virtual ~TimerEvent() {
    if (name != NULL) {
      delete name;
    }
  };
public:
  void setID(Timer::TimerID i) {
    id = i;
  };

  unsigned long getWhen() {
    return when;
  };
  operator const char*() {
    if (name == NULL) {
      char buf[100];
      ::sprintf(buf, "TimerEvent: when=%ld id=%ld", when, id);
      name = new char[strlen(buf) + 1];
      strcpy(name, buf);
    }
    return name;
  };
  virtual bool isRepeating() {
    return FALSE;
  };
  Timer::TimerID getID() {
    return id;
  };
  virtual void run() = 0;
protected:
  friend class Timer;
  unsigned long when;
  Timer::TimerID id;
  char* name;
};
template<class T> class __declspec(dllexport) TimerEventT : public TimerEvent {
public:
  typedef void (T::*RUN)();
  TimerEventT(unsigned long after, T* t, RUN r) : TimerEvent(after + ::GetTickCount()), _this(t), _run(r) {}
  virtual ~TimerEventT(void) {}
  virtual void run(void) {
    (_this->*_run)();
  }
private:
  T* _this;
  RUN _run;
};
template<class T> class __declspec(dllexport) IntervalTimerEventT : public TimerEventT<T> {
  unsigned long interval;
public:
  IntervalTimerEventT(unsigned long i, T* t, RUN r) :
```

```
    TimerEventT<T>(i, t, r), interval(i) {};
  virtual bool isRepeating() {
    when = when + interval;
    return true;
  };
};
/**
 * Global Timer
 */
class __declspec(dllexport) GTimer {
public:
  static Timer::TimerID add(TimerEvent* event);
  static bool remove(Timer::TimerID id);
  static void start(int numberOfThread = 1,
            int priority = THREAD_PRIORITY_NORMAL);
  static void stop();
private:
  static Timer timer;
};
class __declspec(dllexport) StaticIntervalTimerEvent : public TimerEvent {
public:
  void (*runFunc)(void);
  unsigned long interval;
  StaticIntervalTimerEvent(unsigned long pInterval, void (func)(void)) :
              TimerEvent(pInterval + ::GetTickCount()) {
    interval = pInterval;
    runFunc = func;
  }
  virtual bool isRepeating() {
    when = ::GetTickCount() + interval;
    return true;
  };
  virtual void run() {
    (*runFunc)();
  }
};
#endif
```

```
/*
 * TTSChannel.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.12 $
 * $Date: 2000/12/07 03:20:13 $
 */
#include "Channel.h"
#include "VPPUtils/Synchronized.h"
#include "VPPUtils/GMString"
class SampleListener;
class ServiceProvider;
class __declspec(dllexport) TTSChannel : public Channel
{
public:
        virtual bool Shutdown(void);
        virtual ~TTSChannel(void);
        bool InitializeSynthesis(const char *string);
        // return TRUE for full buffer, with padded silence if necessary
        // FALSE - no more data
        int GetNextSamples(char *buffer, unsigned long BufferSize);
        void Release(void);
        virtual void NotifyBoundToSession(SessionNode *pSessionNode);
   void NotifySettled(void);
        typedef enum
        {
                TTSSynthStatus_EOD,
                TTSSynthStatus_NotReady,
                TTSSynthStatus_Ready
        } TTSSynthStatus;
protected:

        typedef enum
        {
                TTS_Invalid,
                TTS_Idle,
                TTS_Initialized,
                TTS_Stop,
                TTS_Exit
        } TTS_ChannelState;
        TTSChannel::TTSChannel(ServiceProvider *pServiceProvider, const char *logType, int id, bool
RequiresSettlement = false);
        virtual bool StartTTS(void) = 0;
        virtual void StopTTS(void) = 0;
        virtual TTSSynthStatus SynthesisStatus(void) = 0;
        virtual int GetNextBuffer(char *buffer, unsigned long m_bufferSize) = 0;
// class data
        TTS_ChannelState        m_State;
        Monitor         m_StateMonitor;
        char            *m_pString;
};
```

```
/*
 * TTSDoneNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:34 $
 */
#ifndef TTSDONENTFY_H
#define TTSDONENTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) TTSDoneNotification : public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(TTSDoneNotification, OneCallTokenMessageBase);
        inline TTSDoneNotification(CallToken& token);
};
inline
TTSDoneNotification::TTSDoneNotification(CallToken& token)
        : OneCallTokenMessageBase(M_NTFY_TTSDONE, token)
{
}
#endif
```

```
/*
 * TTSMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:21:34 $
 */
#ifndef TTSMSG_H
#define TTSMSG_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
#include "VPPUtils/GMString"
#include "VPPUtils/GMLangID.h"
class __declspec(dllexport) TTSMessage : public OneCallTokenMessageBase
{
        gm::string cProductName;
        gm::string cText;
        GMLANGID cLangID;
        unsigned short cSpeed;
        unsigned short cPitch;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(TTSMessage, OneCallTokenMessageBase);
        inline TTSMessage(CallToken& token, const gm::string& text,
                const gm::string& productname, GMLANGID lang, unsigned short Speed,
                unsigned short Pitch);
        inline const gm::string& GetProductName(void);
        inline gm::string& GetText(void);
        inline GMLANGID GetLangID(void);
        inline unsigned short GetSpeed(void);
        inline unsigned short GetPitch(void);
};
inline
TTSMessage::TTSMessage(CallToken& token, const gm::string& text,
        const gm::string& productname, GMLANGID lang, unsigned short Speed,
        unsigned short Pitch) : OneCallTokenMessageBase(M_CMD_TTS, token)
{
        cProductName = productname.c_str();
        cText = text.c_str();
        cLangID = lang;
        cSpeed = Speed;
        cPitch = Pitch;
}
inline const gm::string&
TTSMessage::GetProductName(void)
{
        return cProductName;
}
inline gm::string&
```

```
TTSMessage::GetText(void)
{
        return cText;
}
inline GMLANGID
TTSMessage::GetLangID(void)
{
        return cLangID;
}
inline unsigned short
TTSMessage::GetSpeed(void)
{
        return cSpeed;
}
inline unsigned short
TTSMessage::GetPitch(void)
{
        return cPitch;
}
#endif
```

```
/*
 * TTSPlayingNtfy.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:35 $
 */
#ifndef TTSPLAYINGNTFY_H
#define TTSPLAYINGNTFY_H
#include "OneCallTokenMsgBase.h"
#include "VPPDefs.h"
class __declspec(dllexport) TTSPlayingNotification :
        public OneCallTokenMessageBase
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(TTSPlayingNotification,
                OneCallTokenMessageBase);
        inline TTSPlayingNotification(CallToken& token);
};
inline
TTSPlayingNotification::TTSPlayingNotification(CallToken& token)
        : OneCallTokenMessageBase(M_NTFY_TTSPLAYING, token)
{
}
#endif
```

```
/*
 * TTSSegment.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.7 $
 * $Date: 2000/12/07 03:20:13 $
 */
#ifndef TTSSEGMENT_H
#define TTSSEGMENT_H
#include "PromptSegment.h"
#include <VPPUtils/GMString>
class TTSSegment : public PromptSegment
{
        gm::string cTTSText;
        gm::string cFilename;
public:
        inline TTSSegment(const gm::string& ttsText, const gm::string& filename);
        virtual ~TTSSegment(void);
        virtual bool Play(MediaChannel *pMediaChannel,
                CallToken& ct);
        void Append(const gm::string& ttsText);
};

inline
TTSSegment::TTSSegment(const gm::string& ttsText, const gm::string& filename)
        : PromptSegment(PromptSegment::ST_TTS),
          cTTSText(ttsText.c_str()), cFilename(filename.c_str())
{
}
#endif
```

```
/*
 * TTSServiceProvider.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.6 $
 * $Date: 2000/12/07 03:20:13 $
 */
#ifndef TTSSERVICEPROVIDER_H
#define TTSSERVICEPROVIDER_H
#include "VPPMsg/VPPDefs.h"
#include "ServiceProvider.h"
class MTSServer;
class TTSChannel;
class __declspec(dllexport) TTSServiceProvider : public ServiceProvider
{
protected:
        MTSServer *cServer;
public:
        inline TTSServiceProvider(MTSServer *pServer, const char *logType);
        virtual ~TTSServiceProvider(void);
        virtual bool Setup(void) = 0;
        virtual bool Teardown(void) = 0;
   TTSChannel *FindAvailableChannel(void);
};
inline
TTSServiceProvider::TTSServiceProvider(MTSServer *pServer, const char *logType)
        : ServiceProvider(pServer, M_NTFY_TTSSPFAILURE, logType)
{
        cServer = pServer;
}
typedef TTSServiceProvider *(*CREATETTSSPFUNC)(MTSServer *);
extern "C" __declspec(dllexport)
TTSServiceProvider *CreateTTSServiceProvider(MTSServer *);
#endif
```

```
/*
 * TwoCallTokensMsgBase.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:21:35 $
 */
#ifndef TWOCALLTOKENSMSGBASE_H
#define TWOCALLTOKENSMSGBASE_H
#include "OneCallTokenMsgBase.h"
#include "CallToken.h"
class __declspec(dllexport) TwoCallTokensMessageBase : public OneCallTokenMessageBase
{
        CallToken cCallToken2;
protected:
        virtual void Serialize(void);
        virtual void Deserialize(void);
public:
        DECLARE_CMSMESSAGE_SUBCLASS(TwoCallTokensMessageBase, OneCallTokenMessageBase);
        inline TwoCallTokensMessageBase(unsigned short cmdid, CallToken& CallToken1,
                CallToken& CallToken2);
        inline CallToken& GetCallToken1(void);
        inline CallToken& GetCallToken2(void);
};
inline
TwoCallTokensMessageBase::TwoCallTokensMessageBase(unsigned short cmdid,
        CallToken& CallToken1, CallToken& CallToken2) : OneCallTokenMessageBase(cmdid, CallToken1)
{
        cCallToken2 = CallToken2;
}
inline CallToken&
TwoCallTokensMessageBase::GetCallToken1(void)
{
        return GetCallToken();
}
inline CallToken&
TwoCallTokensMessageBase::GetCallToken2(void)
{
        return cCallToken2;
}
#endif
```

```
/*
 * UtilsLog.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:22:26 $
 */
#ifndef  _UTILS_LOG_H
#define  _UTILS_LOG_H
#include "Log.h"
/*
 *
 */
class UtilsLog : public LogT<UtilsLog> {
   UtilsLog() : LogT<UtilsLog>("VPPUtils") {};
   static UtilsLog log; // for initialization
};
#endif
```

Confidential - Subject to Protective Order

```
/*
 * VPPDebug.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.9 $
 * $Date: 2000/12/07 03:22:26 $
 */
#ifndef VPPDEBUG_H
#define VPPDEBUG_H
#include <stdio.h>
#include <list>
#include "CriticalSection.h"
#include "GMString"
#pragma warning(disable:4786)
class __declspec(dllexport) debug
{
        typedef struct
        {
                char *name;
                bool squelched;
        } DebugNode;
        template class __declspec(dllexport) std::list<DebugNode *>;
        static std::list<DebugNode *> cGroups;
        static FILE *cOutput;
        static FILE *cTeeOutput;
        static CriticalSection cCriticalSection;
        static bool cAtNewline;
        static bool cInitialized;
        static bool cAllPresent;
        static bool cAllSquelched;
        static unsigned long cLastCheck;
        static unsigned long cReparseTime;
        static void Init(void);
        static void ParseDebugGroups(void);
        static void PrintHeader(FILE *fp);
        static int doprint(const char *group, const char *fmt, const va_list vl, bool console);

public:
        static bool IsEnabled(const char *group);
        static void Squelch(const char *group, bool Squelch);

        static void SetOutput(FILE *output);
        static void SetOutput(const char *filename, bool overwrite = false);
        static FILE *GetOutput(void);
        static void SetTeeOutput(FILE *teeoutput);
        static void SetTeeOutput(const char *filename, bool overwrite = false);
        static FILE *GetTeeOutput(void);

        static int printf(const char *group, const char *fmt, ...);
        static int printfandconsole(const char *group, const char *fmt, ...);
        static int vlprintf(const char *group, const char *fmt, const va_list vl);
```

```
        static int console(const char *fmt, ...);
};
#endif
```

Document provided by IHS Inc.

```
/*
 * VPPDefs.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.30 $
 * $Date: 2000/12/07 03:21:35 $
 */
#ifndef VPPDEFS_H
#define VPPDEFS_H
#include "CMS/CMSDefs.h"
/*
 * Typedefs
 */
typedef unsigned short REQUESTID;
#define INVALID_REQUESTID ((REQUESTID)-1)
typedef unsigned short NODEID;
#define INVALID_NODEID ((NODEID)-1)
typedef unsigned short RECORDINGID;
#define INVALID_RECORDINGID ((RECORDINGID)-1)
typedef unsigned short SESSIONID;
#define INVALID_SESSIONID ((SESSIONID)-1)
/*
 * CommandID definitions
 */
#define M_CMDID_START (C_USER_CMDID_START)
#define M_CMD_START (M_CMDID_START)
#define M_RPLY_START (M_CMD_START + 100)
#define M_NTFY_START (M_RPLY_START + 100)
/*
 * Command Messages
 */
#define M_CMD_REGISTERVUI ((unsigned short)(M_CMD_START))
#define M_CMD_UNREGISTERVUI ((unsigned short)(M_CMD_START + 1))
#define M_CMD_PLACECALL ((unsigned short)(M_CMD_START + 2))
#define M_CMD_DROPCALL ((unsigned short)(M_CMD_START + 3))
#define M_CMD_CONNECTCALLS ((unsigned short)(M_CMD_START + 4))
#define M_CMD_LAZYSESSIONSHUTDOWN ((unsigned short)(M_CMD_START + 5))
#define M_CMD_IMMEDIATESESSIONSHUTDOWN ((unsigned short)(M_CMD_START + 6))
#define M_CMD_PROCESSSHUTDOWN ((unsigned short)(M_CMD_START + 7))
#define M_CMD_STOPRECORD ((unsigned short)(M_CMD_START + 8))
#define M_CMD_PLACECALLONHOLD ((unsigned short)(M_CMD_START + 9))
#define M_CMD_ACCEPTCALL ((unsigned short)(M_CMD_START + 10))
#define M_CMD_REJECTCALL ((unsigned short)(M_CMD_START + 11))
#define M_CMD_REGISTERVUIBROKER ((unsigned short)(M_CMD_START + 12))
#define M_CMD_UNREGISTERVUIBROKER ((unsigned short)(M_CMD_START + 13))
#define M_CMD_WRAPPEDMSG ((unsigned short)(M_CMD_START + 14))
#define M_CMD_SETDIALINGPARAMS ((unsigned short)(M_CMD_START + 15))
#define M_CMD_CANCELREQUEST ((unsigned short)(M_CMD_START + 16))
#define M_CMD_DISCONNECTCALLS ((unsigned short)(M_CMD_START + 17))
#define M_CMD_PLAY ((unsigned short)(M_CMD_START + 18))
#define M_CMD_STOPPLAY ((unsigned short)(M_CMD_START + 19))
```

```
#define M_CMD_RECORD ((unsigned short)(M_CMD_START + 20))
#define M_CMD_REMOVERECORDING ((unsigned short)(M_CMD_START + 21))
#define M_CMD_HANDOFFRECORDING ((unsigned short)(M_CMD_START + 22))
#define M_CMD_TTS ((unsigned short)(M_CMD_START + 23))
#define M_CMD_CREATECOMPOSITEPROMPT ((unsigned short)(M_CMD_START + 24))
#define M_CMD_APPENDCOMPOSITEPROMPTPROMPT ((unsigned short)(M_CMD_START + 25))
#define M_CMD_APPENDCOMPOSITEPROMPTSILENCE ((unsigned short)(M_CMD_START + 26))
#define M_CMD_APPENDCOMPOSITEPROMPTTIME ((unsigned short)(M_CMD_START + 27))
#define M_CMD_APPENDCOMPOSITEPROMPTDATE ((unsigned short)(M_CMD_START + 28))
#define M_CMD_APPENDCOMPOSITEPROMPTNUMBER ((unsigned short)(M_CMD_START + 29))
#define M_CMD_APPENDCOMPOSITEPROMPTPHONENUMBER ((unsigned short)(M_CMD_START + 30))
#define M_CMD_APPENDCOMPOSITEPROMPTTTS ((unsigned short)(M_CMD_START + 31))
#define M_CMD_ENDCOMPOSITEPROMPT ((unsigned short)(M_CMD_START + 32))
#define M_CMD_PLAYCOMPOSITEPROMPT ((unsigned short)(M_CMD_START + 33))
#define M_CMD_STARTRECOGNITION ((unsigned short)(M_CMD_START + 34))
#define M_CMD_STOPRECOGNITION ((unsigned short)(M_CMD_START + 35))
#define M_CMD_SETPROPERTY ((unsigned short)(M_CMD_START + 36))
#define M_CMD_GETPROPERTY ((unsigned short)(M_CMD_START + 37))
#define M_CMD_RECOGNIZESTRING ((unsigned short)(M_CMD_START + 38))
#define M_CMD_LOGENTERSTATE ((unsigned short)(M_CMD_START + 39))
#define M_CMD_LOGITEM ((unsigned short)(M_CMD_START + 40))
#define M_CMD_LOGLEAVESTATE ((unsigned short)(M_CMD_START + 41))
#define M_CMD_COMPOSITEMSG ((unsigned short)(M_CMD_START + 42))
#define M_CMD_CREATEDYNAMICGRAMMAR ((unsigned short)(M_CMD_START + 43))
#define M_CMD_DELETEDYNAMICGRAMMAR ((unsigned short)(M_CMD_START + 44))
#define M_CMD_ADDPHRASETODYNAMICGRAMMAR ((unsigned short)(M_CMD_START + 45))
#define M_CMD_REMOVEPHRASEFROMDYNAMICGRAMMAR ((unsigned short)(M_CMD_START + 46))
#define M_CMD_QUERYDYNAMICGRAMMARCONTENTS ((unsigned short)(M_CMD_START + 47))
#define M_CMD_INSERTDYNAMICGRAMMAR ((unsigned short)(M_CMD_START + 48))
#define M_CMD_APPENDCOMPOSITEPROMPTDTMF ((unsigned short)(M_CMD_START + 49))
/*
 * Reply Messages (for transacted commands)
 */
#define M_RPLY_REGISTERVUI ((unsigned short)(M_RPLY_START))
#define M_RPLY_REGISTERVUIBROKER ((unsigned short)(M_RPLY_START + 3))
/*
 * Notification Messages
 */
#define M_NTFY_SUSPENDED ((unsigned short)(M_NTFY_START))
#define M_NTFY_SESSIONSHUTDOWNCOMPLETE ((unsigned short)(M_NTFY_START + 1))
#define M_NTFY_PROCESSSHUTDOWNCOMPLETE ((unsigned short)(M_NTFY_START + 2))
#define M_NTFY_CALLPROGRESS ((unsigned short)(M_NTFY_START + 3))
#define M_NTFY_CALLDROPPED ((unsigned short)(M_NTFY_START + 4))
#define M_NTFY_CALLSCONNECTED ((unsigned short)(M_NTFY_START + 5))
#define M_NTFY_CALLPLACEDONHOLD ((unsigned short)(M_NTFY_START + 6))
#define M_NTFY_INCOMINGCALL ((unsigned short)(M_NTFY_START + 7))
#define M_NTFY_ASSIGNEDCLIENTID ((unsigned short)(M_NTFY_START + 8))
#define M_NTFY_CALLCONNECTED ((unsigned short)(M_NTFY_START + 9))
#define M_NTFY_REQUESTCANCELLED ((unsigned short)(M_NTFY_START + 10))
#define M_NTFY_SITDETECTED ((unsigned short)(M_NTFY_START + 11))
#define M_NTFY_CNGDETECTED ((unsigned short)(M_NTFY_START + 12))
#define M_NTFY_CALLSDISCONNECTED ((unsigned short)(M_NTFY_START + 13))
#define M_NTFY_PLAYING ((unsigned short)(M_NTFY_START + 14))
#define M_NTFY_PLAYDONE ((unsigned short)(M_NTFY_START + 15))
#define M_NTFY_STOPPEDPLAYING ((unsigned short)(M_NTFY_START + 16))
#define M_NTFY_RECORDING ((unsigned short)(M_NTFY_START + 17))
```

#define M_NTFY_TSPFAILURE ((unsigned short)(M_NTFY_START + 18))
#define M_NTFY_MSPFAILURE ((unsigned short)(M_NTFY_START + 19))
#define M_NTFY_RECORDINGHANDEDOFF ((unsigned short)(M_NTFY_START + 20))
#define M_NTFY_TTSPLAYING ((unsigned short)(M_NTFY_START + 21))
#define M_NTFY_TTSDONE ((unsigned short)(M_NTFY_START + 22))
#define M_NTFY_COMPOSITEPROMPTPLAYING ((unsigned short)(M_NTFY_START + 23))
#define M_NTFY_COMPOSITEPROMPTSEGMENTPLAYING ((unsigned short)(M_NTFY_START + 24))
#define M_NTFY_COMPOSITEPROMPTSEGMENTPLAYDONE ((unsigned short)(M_NTFY_START + 25))
#define M_NTFY_COMPOSITEPROMPTPLAYDONE ((unsigned short)(M_NTFY_START + 26))
#define M_NTFY_COMPOSITEPROMPTCREATED ((unsigned short)(M_NTFY_START + 27))
#define M_NTFY_COMPOSITEPROMPTSEGMENTAPPENDED ((unsigned short)(M_NTFY_START + 28))
#define M_NTFY_COMPOSITEPROMPTENDED ((unsigned short)(M_NTFY_START + 29))
#define M_NTFY_RESUME ((unsigned short)(M_NTFY_START + 30))
#define M_NTFY_STOPPEDRECORDING ((unsigned short)(M_NTFY_START + 31))
#define M_NTFY_INCOMINGCALLREVOKED ((unsigned short)(M_NTFY_START + 32))
#define M_NTFY_SSPFAILURE ((unsigned short)(M_NTFY_START + 33))
#define M_NTFY_RECOGNITIONDONE ((unsigned short)(M_NTFY_START + 34))
#define M_NTFY_PROPERTYSET ((unsigned short)(M_NTFY_START + 35))
#define M_NTFY_PROPERTYGOTTEN ((unsigned short)(M_NTFY_START + 36))
#define M_NTFY_PROPERTYNOTSET ((unsigned short)(M_NTFY_START + 37))
#define M_NTFY_PROPERTYNOTGOTTEN ((unsigned short)(M_NTFY_START + 38))
#define M_NTFY_BEGINSPEECH ((unsigned short)(M_NTFY_START + 39))
#define M_NTFY_ENDSPEECH ((unsigned short)(M_NTFY_START + 40))
#define M_NTFY_TTSSPFAILURE ((unsigned short)(M_NTFY_START + 41))
#define M_NTFY_DTMF ((unsigned short)(M_NTFY_START + 42))
#define M_NTFY_CALLACCEPTED ((unsigned short)(M_NTFY_START + 43))
#define M_NTFY_CALLREJECTED ((unsigned short)(M_NTFY_START + 44))
#define M_NTFY_RECOGNITIONSTARTED ((unsigned short)(M_NTFY_START + 45))
#define M_NTFY_MTSLINKDOWN ((unsigned short)(M_NTFY_START + 46))
#define M_NTFY_CDRINFO ((unsigned short)(M_NTFY_START + 47))
#define M_NTFY_OUTBOUNDCHANNELAVAILABLE ((unsigned short)(M_NTFY_START + 48))
#define M_NTFY_MTSLINKUP ((unsigned short)(M_NTFY_START + 49))
#define M_NTFY_VUIBROKERLINKDOWN ((unsigned short)(M_NTFY_START + 50))
#define M_NTFY_VUIBROKERLINKUP ((unsigned short)(M_NTFY_START + 51))
#define M_NTFY_MESSAGINGFAILURE ((unsigned short)(M_NTFY_START + 52))
#define M_NTFY_INBOUNDCALLCAPACITY ((unsigned short)(M_NTFY_START + 53))
#define M_NTFY_T2S_SERVER_BEGIN_CONN ((unsigned short)(M_NTFY_START + 54))
#define M_NTFY_T2S_SERVER_CLOSE_CONN ((unsigned short)(M_NTFY_START + 55)
#define M_NTFY_T2S_SERVER_END_CONN ((unsigned short)(M_NTFY_START + 56))
#define M_NTFY_T2S_SERVER_TTS_REQUEST ((unsigned short)(M_NTFY_START + 57))
#define M_NTFY_T2S_SERVER_TTS_AUDIO_DATA_SEG ((unsigned short)(M_NTFY_START + 58))
#define M_NTFY_T2S_SERVER_TTS_COMPLETE ((unsigned short)(M_NTFY_START + 59))
#define M_NTFY_T2S_SERVER_COMM_ERROR ((unsigned short)(M_NTFY_START + 60))
#define M_NTFY_T2S_SERVER_TTS_MESSAGE ((unsigned short)(M_NTFY_START + 61))
#define M_NTFY_T2S_SERVER_TTS_INITIALIZE ((unsigned short)(M_NTFY_START + 62))
#define M_NTFY_T2S_SERVER_TTS_START ((unsigned short)(M_NTFY_START + 63))
#define M_NTFY_T2S_SERVER_TTS_STOP ((unsigned short)(M_NTFY_START + 64))
#define M_NTFY_T2S_SERVER_TTS_EXIT ((unsigned short)(M_NTFY_START + 65))
#define M_NTFY_CREATEDYNAMICGRAMMARRESULT ((unsigned short)(M_NTFY_START + 66))
#define M_NTFY_DELETEDYNAMICGRAMMARRESULT ((unsigned short)(M_NTFY_START + 67))
#define M_NTFY_ADDPHRASETODYNAMICGRAMMARRESULT ((unsigned short)(M_NTFY_START + 68))
#define M_NTFY_REMOVEPHRASEFROMDYNAMICGRAMMARRESULT ((unsigned short)(M_NTFY_START + 69))
#define M_NTFY_QUERYDYNAMICGRAMMARCONTENTSRESULT ((unsigned short)(M_NTFY_START + 70))

```
#define M_NTFY_INSERTDYNAMICGRAMMARRESULT ((unsigned short)(M_NTFY_START + 71))
/*
 * Info. codes for TSP Failure Notifications
 */
#define TSPF_OK                          (0x00)
#define TSPF_CANTGOOFFHOOK               (0x01)
#define TSPF_CANTGOONHOOK                (0x02)
#define TSPF_INVALIDCALLTOKEN            (0x03)
#define TSPF_NOAVAILABLECHANNEL          (0x04)
#define TSPF_INVALIDREQUESTTOKEN         (0x05)
#define TSPF_CANTDIAL                    (0x06)
#define TSPF_CANTCANCEL                  (0x07)
#define TSPF_CANTLISTEN                  (0x08)
#define TSPF_CALLNOTOFFERED              (0x09)
#define TSPF_CANTANSWER                  (0x0A)
#define TSPF_CANTGETTIMESLOT             (0x0B)
#define TSPF_CANTCREATESESSION           (0x0C)
#define TSPF_INSUFFICIENTRESOURCES       (0x0D)
#define TSPF_NOSERVICEPROVIDER           (0x0E)
#define TSPF_SYSTEMFAILURE               (0x0F)
#define TSPF_CALLALREADYDISCONNECTED     (0x10)
#define TSPF_RESTRICTEDNUMBER            (0x11)
/*
 * Info. codes for MSP Failure Notifications
 */
#define MSPF_OK                          (0x00)
#define MSPF_NOTIDLE                     (0x01)
#define MSPF_CANTSTOP                    (0x02)
#define MSPF_CANTFINDPROMPT              (0x03)
#define MSPF_CANTRECORD                  (0x04)
#define MSPF_CANTPLAY                    (0x05)
#define MSPF_INVALIDCALLTOKEN            (0x06)
#define MSPF_NOTHINGTOSTOP               (0x07)
#define MSPF_NOTHINGTOPLAY               (0x08)
#define MSPF_NOSUCHRECORDING             (0x09)
#define MSPF_CANTHANDOFFRECORDING        (0x0A)
#define MSPF_TTSSYSTEMFAILURE            (0x0B)
#define MSPF_CANTTTS                     (0x0C)
#define MSPF_INSUFFICIENTRESOURCES       (0x0D)
#define MSPF_NOSERVICEPROVIDER           (0x0E)
#define MSPF_CANTPLAYPROMPTSEGMENT       (0x0F)
#define MSPF_COMPOSITEPROMPTFAILURE      (0x10)
/*
 * Info. codes for SSP Failure Notifications
 */
#define SSPF_OK                          (0x00)
#define SSPF_INVALIDCALLTOKEN            (0x01)
#define SSPF_NOTIDLE                     (0x02)
#define SSPF_NOLICENSE                   (0x03)
#define SSPF_CANTRECOGNIZE               (0x04)
#define SSPF_RECORDCHANNELNOTIDLE        (0x05)
#define SSPF_CANTRECORD                  (0x06)
#define SSPF_INSUFFICIENTRESOURCES       (0x07)
#define SSPF_NOSERVICEPROVIDER           (0x08)
#define SSPF_UNKNOWNGRAMMAR              (0x09)
#define SSPF_DATABASEERROR               (0x0A)
```

#define SSPF_SYSTEMFAILURE (0x0B)
#endif

Downloaded from

```
/*
 * VPPMsg.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:36 $
 */
#ifndef VPPMSG_H
#define VPPMSG_H
#include "CMS/CMSMsg.h"
#if 0
class VirtualMachineID;
#endif
class __declspec(dllexport) VPPMessage : public CMSMessage
{
public:
        DECLARE_CMSMESSAGE_SUBCLASS(VPPMessage, CMSMessage);
        inline VPPMessage(void);
        inline VPPMessage(unsigned short cmdid);
        inline VPPMessage(const VPPMessage& other);
        inline VPPMessage(VPPMessage *pOther);
#if 0
        virtual VirtualMachineID& GetMTSVirtualMachineID(void);
#endif
};
inline
VPPMessage::VPPMessage(void)
        : CMSMessage()
{
}
inline
VPPMessage::VPPMessage(unsigned short cmdid)
        : CMSMessage(cmdid)
{
}
inline
VPPMessage::VPPMessage(const VPPMessage& other)
        : CMSMessage(other)
{
}
inline
VPPMessage::VPPMessage(VPPMessage *pOther)
        : CMSMessage(pOther)
{
}
#endif
```

```
/*
 * VPPServer.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:37 $
 */
#ifndef VPPSERVER_H
#define VPPSERVER_H
#include "CMS/CMSServer.h"
class __declspec(dllexport) VPPServer : public CMSServer
{
public:
        inline VPPServer(IPCEndpoint *pIPCEndpoint = 0);
};
inline
VPPServer::VPPServer(IPCEndpoint *pIPCEndpoint)
        : CMSServer(pIPCEndpoint)
{
}
#endif
```

```
/*
 * VRSAPIClient.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.27 $
 * $Date: 2000/12/07 03:24:29 $
 */
#ifndef VRSAPIUSER_H
#define VRSAPIUSER_H
//#define NUANCE7
#define _WIN32_WINNT 0x0400
#include "audio-format.h"
#ifdef NUANCE7
#include "dispatcher-interfaces.hpp"
#include "event-notifiable.hpp"
#else
#include "dispatcher-interfaces.hh"
#include "event-notifiable.hh"
#endif
#include "recresult.h"
#include "nuance-config.h"
#include "db-descriptor.h"
#include "VPPUtils/GMString"
#include "VPPUtils/Log.h"
#include "VPPMsg/DynamicGrammarResultNtfy.h"
#include <list>
class NuanceSChannel;
class MediaChannel;
class Dispatcher;
class VRecServer;
class ConfigFile;
class RecognitionResult;
class Slot;
class Property;
class SessionNode;
class VRSAPIClient : public EventNotifiable
{
        friend class AbortUttTimer;
public:
        typedef enum
        {
                S_NotInitialized,
                S_Idle,
                S_Recognizing,
                S_WaitingForResult
        } RecognitionState;
private:
        typedef struct
        {
                unsigned int cID;
                gm::string cDBKey;
```

```
            gm::string cPhraseID;
            bool cAddition;
            gm::string cGrammarName;
        } RequestNode;
        AudioFormat cAudioFormat;
        VRecServer *cVRecServer;
        NuanceSChannel *cSpeechChannel;
        unsigned int cID;
        RecognitionState cState;
        Dispatcher *cDispatcher;
        unsigned long cRunThread;
        gm::string cUtteranceFilename;
        unsigned long cRecognitionStartTime;
        unsigned long cRecognitionStoppedEvt;
        unsigned long cDispatcherThreadStoppedEvt;
        DBDescriptor *cDB;
        unsigned long cDatabaseOpenedEvt;
        unsigned int cDBID;
        unsigned int cRequestID;
        template class __declspec(dllexport) std::list<RequestNode *>;
        std::list<RequestNode *> cRequests;
        static unsigned int cNextDBID;
        static char *cDatabaseKeys[];

        bool OpenDatabaseConnection(void);
        bool CloseDatabaseConnection(void);
        void InitializationDone(NuanceStatus status);
        void StartOfSpeech(unsigned id, int safe_index, int actual_index);
        void EndOfSpeech(unsigned id, int index);
        void FinalResult(unsigned id, RecResult const *rrp);
        void FinalInterpretation(NLResult const *pNLResult);
        void HandleException(NuanceStatus status);
        void HandleError(RecognitionState state, Notification::Type type);
        void DynamicGrammarCreated(NuanceStatus status, unsigned int id);
        void DynamicGrammarDeleted(NuanceStatus status, unsigned int id);
        void DynamicGrammarModified(NuanceStatus status, unsigned int id);
        void DynamicGrammarContents(NuanceStatus status, unsigned int id, const char *phrase_contents);
        void DynamicGrammarInserted(NuanceStatus status, unsigned int id);

        static unsigned __stdcall _RunThreadFunc(void *);
        unsigned RunThreadFunc(void);
        void ResetState(void);
        bool FillSlot(Slot *pSlot, NLValue *pNLValue, gm::string& grammar_key);
        RecognitionResult *ComposeRecognitionResult(RecResult const *rrp, int answer,
            NLResult const *pNLResult);
        unsigned int AddRequestNode(const gm::string& db_key,
            const gm::string& phrase_id = gm::string::Nullref(),
            bool addition = false, const gm::string& grammar_name = gm::string::Nullref());
        bool RemoveRequestNode(unsigned int id);
        RequestNode *FindRequestNode(unsigned int id);
#if 0
        DynamicGrammarResultNotification::Result MapNuanceStatus(NuanceStatus stat);
#endif
public:
        VRSAPIClient(NuanceSChannel *pSpeechChannel);
        ~VRSAPIClient(void);
```

```
        bool Setup(unsigned int id, NuanceConfig *config, AudioFormat *pAudioFormat,
                NuanceStatus& status);
        bool Teardown(void);
        void NUANCE_MEMBER_FUNCTION Notify(EventNotification *&en);
        unsigned long StartRecognition(const gm::string& grammar);
        unsigned long PushSamples(const unsigned char *pSamples, unsigned int numbytes);
        unsigned long StopRecognition(void);
        unsigned long AbortRecognition(void);
        unsigned long RecognizeString(const gm::string& grammar, const gm::string& str);
        unsigned long CreateDynamicGrammar(const gm::string& db_key, const gm::string& gsl);
        unsigned long DeleteDynamicGrammar(const gm::string& db_key);
        unsigned long AddPhraseToDynamicGrammar(const gm::string& db_key,
                const gm::string& phrase_id, const gm::string& phrase_text,
                const gm::string& phrase_nl, const unsigned long probability);
        unsigned long RemovePhraseFromDynamicGrammar(const gm::string& db_key,
                const gm::string& phrase_id);
        unsigned long QueryDynamicGrammarContents(const gm::string& db_key,
                const gm::string& phrase_id);
        unsigned long InsertDynamicGrammar(const gm::string& db_key,
                const gm::string& grammar_name);
        void NotifyBoundToSession(SessionNode *pSessionNode);
        bool StartLogging(void);
        bool LogUtteranceFilename(void);
        bool StopLogging(void);
        void NotifyPropertySet(Property& prop);
        void GetUtteranceFilename(gm::string& filename);
        void AttachRecordingFilename(const gm::string& filename);
};
class AbortUttTimer : public Triggerable
{
public:
        AbortUttTimer(DispatcherInterface& dispatcher, float interval_ms, VRSAPIClient& vrsapi_client);
        void Respond(Trigger const & t, DispatchEntryHandle de);
protected:
        ~AbortUttTimer(void) {}
private:
        DispatcherInterface& cDisp; // Dispatcher
        DispatchEntryHandle cTimeTriggerDispEntry; // timer trigger dispatcher entry
        VRSAPIClient& cVRSAPIClient;
};
#endif
```

```
/*
 * VUISessionImpl.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.13 $
 * $Date: 2000/12/08 01:53:05 $
 */
#ifndef _VUISESSIONIMPL_H
#define _VUISESSIONIMPL_H
#include "VPPUtils/Hashtable.h"
#include "invui_s.h"
#include "SessionNode.h"
#include "VppMsg/RecordingDoneNtfy.h"
class __declspec(dllexport) VUISessionImpl : public SessionNode,
                                             public ConnectionListener,
                                             public POA_invui::VUISession
{
  invui::VUIClient_var cVUIClient;
  unsigned long cMediaStatus;
  unsigned long cSpeechStatus;
  unsigned long cTelephonyStatus;
  /**
   * We may be able to use MediaChannel::ChannelState
   */
  enum {
    PROMPT_NONE,
    PROMPT_PLAYING,
    PROMPT_DONE
  } cPromptState;
  /**
   * We may be able to use SpeechChannel::ChannelState
   */
  enum {
    RECOGNITION_NONE,
    RECOGNITION_STARTED,
    RECOGNITION_DONE,
  } cRecognitionState;
  enum {
    RECORDING_NONE,
    RECORDING_STARTED,
    RECORDING_DONE
  } cRecordingState;
  enum {
    ODS_NONE,
    ODS_ALERTING_NOTIFY,
    ODS_ALERTING,
    ODS_CONNECTED
  } cOutdialState;
  invui::PlayAndRecResults cPlayAndRecResults;
  invui::RecordResult cRecordResult;
  virtual void Disconnected();
```

```
  bool CheckValid();
  bool CheckMedia();
  bool CheckSpeech();
  bool CheckTelephony();
  void StartPrompts();
  /*************************************************************/
  //
  // Notification METHODS
  //
  /*************************************************************/
  virtual CMSMessage::SerialNumberT NotifyMSPFailure(CallToken& ct, CMSMessage::SerialNumberT
FailureSerialNo, unsigned long Info);
  virtual CMSMessage::SerialNumberT NotifySSPFailure(CallToken& ct, CMSMessage::SerialNumberT
FailureSerialNo, unsigned long Info);
  virtual CMSMessage::SerialNumberT NotifyTSPFailure(CallToken& ct, CMSMessage::SerialNumberT
FailureSerialNo, unsigned long Info);
  virtual CMSMessage::SerialNumberT NotifyTSPFailure(CMSMessage::SerialNumberT FailureSerialNo, unsigned
long Info);
  virtual CMSMessage::SerialNumberT NotifyCallProgress(RequestToken& request,
                                CallProgressNotification::CallProgressT notification);
  virtual CMSMessage::SerialNumberT NotifyCallConnected(RequestToken& request, CallToken& callToken);
  virtual CMSMessage::SerialNumberT NotifyCallDropped(CallToken& ct);
  virtual void NotifyIncomingCallRevoked(CallToken& ct);
  virtual CMSMessage::SerialNumberT NotifyCompositePromptPlayDone(CallToken&
callToken);
  virtual CMSMessage::SerialNumberT NotifyRecording(CallToken& callToken, RECORDINGID recordingID, const
gm::string& filename);
  virtual CMSMessage::SerialNumberT NotifyRecordingDone(CallToken& callToken,
RecordingDoneNotification::RecordingDoneReason reason, unsigned long duration);
  virtual CMSMessage::SerialNumberT NotifyCompositePromptSegmentPlayDone(CallToken& callToken);
  virtual CMSMessage::SerialNumberT NotifyRecognitionDone(CallToken& ct,
                                RecognitionResults *pResults,
                                RecognitionDoneNotification::RecognitionStatus status);
  virtual CMSMessage::SerialNumberT NotifyDTMF(CallToken& callToken,
                            char dtmf, unsigned long timestamp) ;
  /*************************************************************/
  //
  // CORBA METHODS
  //
  /*************************************************************/
  virtual void clearPrompts();
  virtual invui::PlayAndRecResults* playAndRecognize(const invui::GrammarHandle& handle,
                                invui::TIME bos_timeout,
                                invui::TIME tms_timeout,
                                invui::TIME eos_timeout);
  virtual char* getProperty(const char* key);
  virtual CORBA::Boolean stopPlay(CORBA::Boolean wait_for_segment);
  virtual CORBA::Boolean cancelPlay();
  virtual void appendPrompts(const invui::Prompts& ps);
  virtual void addToOutput(const invui::Call& c);
  virtual invui::GrammarHandle* getGrammarHandle(const invui::DynamicGrammar& grammar);
  virtual void removeFromInput(const invui::Call& c);
  virtual void removeFromOutput(const invui::Call& c) ;
  virtual invui::RecognitionResults* recognize(const invui::GrammarHandle& handle,
                            invui::TIME bos_timeout,
                            invui::TIME tms_timeout,
```

```
                                invui::TIME eos_timeout);
    virtual invui::Calls* all_calls();
    virtual void setProperties(const invui::Properties& params);
    virtual invui::TelephonyControl_ptr getTelephonyControl();
    virtual invui::VUISession_ptr changeOwner(invui::VUIClient_ptr onwer);
    virtual void addToInput(const invui::Call& c);
    virtual invui::RecordResult* record(const char* filename,
                                invui::TIME timeout,
                                invui::TIME tms_timeout,
                                invui::TIME eos_timeout);
    virtual CORBA::Boolean cancelRecord();
    virtual CORBA::Boolean cancelRecognize();
    virtual void setProperty(const char* key, const char* value);
    virtual void setSearchPath(const char* _path);
    /*
     * TelephonyControl methods. Those are inherited from TelephonyControl
     */
    virtual CORBA::Boolean acceptCall(const invui::Call& call);
    virtual CORBA::Boolean cancelCall(const invui::Call& call);
    virtual invui::Call* transfer(const char* _number, invui::TIME timeout);
    virtual void dropCalls();
    virtual void dropCall(const invui::Call& c1);
    virtual invui::Call* placeCall(const char* number,
                    const char* ani, const invui::UUIType& uui, invui::TIME timeout);
    virtual void connectCalls(const invui::Call& c1, const invui::Call& c2);
    virtual void disconnectCalls(const invui::Call& c1, const invui::Call& c2);
    virtual void appendPrompt(const invui::Prompt& p);
    virtual invui::PlayResult playPrompts();
    virtual invui::PlayResult playPrompts2(CORBA::Boolean _async);
    /*
     * For Logging. Those may be removed later.
     */
    virtual void enterState(const char* _name);
    virtual void leaveState();
    virtual void log(const char* _name, const char* _value);
    CallToken& GetCurrentCallToken();
private:
friend
    CallToken& GetCurrentCallToken();
public:
    VUISessionImpl(SessionMgr *pSessionMgr, TelephonyChannel *pTChannel);
    virtual ~VUISessionImpl();
    void SetVUIClient(invui::VUIClient_var client);
    virtual void rejectCall(const invui::Call& call);
    invui::VUISession_ptr duplicateReference();
    void deactivate();
    inline void SetCallAlerting() {
      cOutdialState = ODS_ALERTING;
    }
    virtual void terminate();
    void AppendPrompt(const invui::Prompt& p) {  appendPrompt(p); };
    invui::PlayResult PlayPrompts() { return playPrompts(); };
};
#endif
```

```
/*
 * WFStreamer.h
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.10 $
 * $Date: 2000/12/07 03:20:15 $
 */
#ifndef WFSTREAMER_H
#define WFSTREAMER_H
#define ENABLE_PERF_MEASURE
#include <stdio.h>
#include <windows.h>
#include <process.h>
#include "SampleTransmitter.h"
class WFStreamer : public SampleTransmitter
{
public:
        enum Status {
                eS_OK,
                eS_NotOK
        };
        enum State {
                eS_Streaming,
                eS_Idle
        };
        WFStreamer(const char *filename, int bufSize = 500, int transmissionInterval = 0); // bufSize in bytes,
transmissionInterval in msecs
        ~WFStreamer(void);
        HANDLE Go();
        void NotifyFileWriteStopped(void); // used to notify this object that the file it's reading has stopped growing
        bool IsReadyToTransmit(void);
        bool StartTransmission(void);
        bool StopTransmission(void);
        bool WaitTillIdle(void);
        void SetDeleteFile(bool bDelete);
        inline bool SetSampleListener(SampleListener *pListener);
#ifdef ENABLE_PERF_MEASURE
        inline void SetStartRecordingTime(unsigned long time);
#endif
private:
        FILE *cpFile;
        char *cpFilename;
        char *cpBuf;
        int  cBufSize;
        fpos_t cCurFilePos;
        fpos_t cEOFPos;
        bool   cHeaderRead;
        Status cStatus;     // used to check if object construction is ok
        State  cState;
        bool cDeleteFile;
        int  cNumDeleteTries;
```

```
        SampleListener *cpSampleListener;
        HANDLE cStopTransmissionEvent;
        bool cFileWriteStopped;
        int  cTransIntervalmsecs;
        // thread-related stuff
        HANDLE cThreadH;
        DWORD  cThreadId;
        void Read(void);
        FILE *OpenFile(const char *filename, const char *mode);
        void Reset(void);
        static unsigned __stdcall goThreadFunc(LPVOID);
        unsigned GoThreadFunc(void);
#ifdef ENABLE_PERF_MEASURE
        bool cFirstSample;
        unsigned long cStartRecordingTime;
        unsigned long cPushSamplesTime;
#endif
};
inline bool
WFStreamer::SetSampleListener(SampleListener *pListener)
{
        cpSampleListener = pListener;
        return true;
}
#ifdef ENABLE_PERF_MEASURE
inline void
WFStreamer::SetStartRecordingTime(unsigned long time)
{
        cStartRecordingTime = time;
}
#endif
#endif
```

```
/*
 * AcceptCallMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:21:44 $
 */
static const char* id="$Id: AcceptCallMsg.cpp,v 1.4 2000/12/07 03:21:44 Mitsuru Oshima Exp $";
#include "AcceptCallMsg.h"
#include "Property.h"
#include "PropertyList.h"
// virutal
AcceptCallMessage::~AcceptCallMessage(void)
{
        if (cOwned)
                delete cProperties;
}
// virtual
void
AcceptCallMessage::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        cProperties->Serialize(this);
        cCDRInfo.Serialize(this);
}
// virtual
void
AcceptCallMessage::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        cOwned = true;
        cProperties = new PropertyList;
        cProperties->Deserialize(this);
        cCDRInfo.Deserialize(this);
}
```

```
/*
 * AcuvoiceTTSChannel.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.18 $
 * $Date: 2000/12/07 03:34:26 $
 */
static const char* id="$Id: AcuvoiceTTSChannel.cpp,v 1.18 2000/12/07 03:34:26 Mitsuru Oshima Exp $";
#include <assert.h>
#include <windows.h>
#include <process.h>
#include <stdio.h>
#include "VPPUtils/VPPDebug.h"
#include "VPPUtils/Synchronized.h"
#include "VPPUtils/GMString"
#include "AcuvoiceTTSChannel.h"
#include "VPPUtils/ConfigMgr.h"
static void _TTSChannelThread(void *);
// the following methods run in main thread
AcuvoiceTTSChannel::AcuvoiceTTSChannel(ServiceProvider *pServiceProvider,
        const char *cLogType, int id)
   : TTSChannel(pServiceProvider, cLogType, id, true),
        synSize(0), dataAvail(0), synthisizing(false),
        audioBufOutRemaining(0), startedSynth(false), m_ThreadHandle(-1)
{
        audioBuffer = new char[AccuvoiceBufSize + MarginDataSize];
        m_EventSignalHandle = ::CreateEvent( NULL, false, false, NULL);

        if(m_EventSignalHandle)
        {
                StartWorkerThread();
        }
        else
        {
                error("Failed to Create Signal Event");
                return;
        }
}
bool
AcuvoiceTTSChannel::Shutdown(void)
{
        return true;
}
AcuvoiceTTSChannel::~AcuvoiceTTSChannel()
{
        // signal worker thread to exit
        SendTtsRequest(TTSRequest_Exit);
        // make sure that worker thread is out
        if (m_ThreadHandle != -1)
        {
                ::WaitForSingleObject((HANDLE)m_ThreadHandle, 200);
```

```
	}
	if(m_EventSignalHandle)
	{
		::CloseHandle(m_EventSignalHandle);
	}
	delete [] audioBuffer;
}
void AcuvoiceTTSChannel::StartWorkerThread()
{
	assert(m_ThreadHandle == -1);
	m_ThreadHandle = ::_beginthread(_TTSChannelThread, 0, this);
	if (m_ThreadHandle != -1)
	{
		gm::string lowPriority;
		// Set thread to lower priority, if the registry tells us to.
		if (ConfigMgr::GetString(MTS_REGISTRY_BASE ".TTS Service Providers.Acuvoice",
			"UseLowPriorityThread", lowPriority) && (::atoi(lowPriority.c_str()) != 0)) {
			::SetThreadPriority((HANDLE)m_ThreadHandle,
THREAD_PRIORITY_BELOW_NORMAL );
		}
		m_State = TTS_Idle;
	}
	else
	{
		error("Failed to start Channel TTS Thread");
	}
}
bool AcuvoiceTTSChannel::StartTTS(void)
{
	diag("StartTTS()");
	audioBufInPtr = audioBuffer;
	audioBufOutPtr = audioBuffer;
	synSize = 0;
	audioBufOutRemaining = 0;
	dataAvail = 1;
	if (synthisizing)
		error("StartTTS request when already synthisizing");
	synthisizing = true;
	SendTtsRequest(TTSRequest_Start);
	return true;
}
bool AcuvoiceTTSChannel::NextTTS(void)
{
	diag("NextTTS()");
	Synchronized sync(m_Monitor);
	if (synthisizing)
	{
		error("NextTTS request when already synthisizing");
		return false;
	}
	synthisizing = true;
	synSize = 0;
	SendTtsRequest(TTSRequest_Next);
	return true;
}
void AcuvoiceTTSChannel::StopTTS(void)
```

```
{
        diag("StopTTS()");
        Synchronized sync(m_Monitor);
        synSize = 0;
        dataAvail = 0;
        audioBufOutRemaining = 0;
        SendTtsRequest(TTSRequest_Stop);
        synthisizing = false;
}
int AcuvoiceTTSChannel::GetNextBuffer(char *buffer, unsigned long m_bufferSize)
{
        debug("GetNextBuffer()");
        CheckBuffer();
        Synchronized sync(m_Monitor);    // Sync needed only if more than one thread calls TTSChannel functions.
        int returnDataSize = (m_bufferSize < audioBufOutRemaining) ? m_bufferSize : audioBufOutRemaining;
        if (returnDataSize != 0)
        {
                memcpy(buffer, audioBufOutPtr, returnDataSize);
                audioBufOutRemaining -= returnDataSize;
                audioBufOutPtr += returnDataSize;
        }
        CheckBuffer();

        return(returnDataSize);
}
TTSChannel::TTSSynthStatus
AcuvoiceTTSChannel::SynthesisStatus(void)
{
        CheckBuffer();
        if (audioBufOutRemaining != 0)
                return TTSSynthStatus_Ready;
        else if (dataAvail || synSize)
                return TTSSynthStatus_NotReady;
        else
                return TTSSynthStatus_EOD;
}
void AcuvoiceTTSChannel::CheckBuffer(void)
{
        Synchronized sync(m_Monitor);    // Sync needed only if more than one thread calls TTSChannel functions.
        if(synthisizing)
        {
                if (synSize != 0)  // New datablock in audioBuffer
                {
                        Synchronized sync(m_WorkerMonitor);
                        audioBufOutRemaining += synSize;
                        synSize = 0;
                        synthisizing = false;
                }
                else if (!dataAvail)
                {
                        synthisizing = false;
                }
    else if (audioBufOutRemaining == 0)
    {
      // Should happen only at beginning.  Otherwise,
      // means that we did not have enough time to
```

```
        // produce new data block
        debug("Underflow, audioBuffer is not ready!");
    }
        }
        if(!synthisizing && dataAvail && (audioBufOutRemaining <= MarginDataSize))
        {
                // Move what's left to beginning of audioBuffer
                if ((audioBufOutRemaining != 0) && (audioBuffer != audioBufOutPtr))
                {
                        memcpy(audioBuffer, audioBufOutPtr, audioBufOutRemaining);
                }
                audioBufOutPtr = audioBuffer;
                audioBufInPtr = audioBufOutPtr + audioBufOutRemaining;
                NextTTS();
        }
}
bool AcuvoiceTTSChannel::SendTtsRequest(TTSRequest request)
{
        bool rc = false;
        Synchronized sync(m_WorkerMonitor);
        if(m_EventSignalHandle != NULL && m_ThreadHandle != -1)
        {
                ttsRequest = request;
                if(::SetEvent(m_EventSignalHandle))
                {
                        rc = true;
                }
                else
                {
                        error("TTSProcess() - SetEvent failed %d", GetLastError());
                        m_State = TTS_Exit;
                }
        }
        return rc;
}
//-------------------------------------------------------------
//
//      The following methods run in worker thread
//
//-------------------------------------------------------------
void AcuvoiceTTSChannel::_TTSChannelThread(void *ptr)
{
        ((AcuvoiceTTSChannel *)ptr)->TTSProcess();
}
void AcuvoiceTTSChannel::TTSProcess(void)
{
        bool done = false;
        DWORD waitRc;
        TTS_ChannelState request;
        TTSRequest lastRequest = TTSRequest_Stop;
        info("Start TTSProcess() Thread");
#ifdef  AVCORE
        allocate_SSB(&m_SSBPtr);
#else
        m_SSBPtr = (SSB_PTR) new char[sizeof(SSB)];
#endif
```

```
        if ( m_SSBPtr == NULL )
        {
                printf("Cannot Allocate SSB Variable.\n");
                error("Cannot Allocate SSB Variable.");
                return;
        }
        while(!done)
        {
                waitRc = ::WaitForSingleObject(m_EventSignalHandle, INFINITE);
                if(waitRc != WAIT_OBJECT_0)
                {
                        done = true;
                        error("TTSProcess() WaitForSingleObject failed %d",
                                GetLastError());
                        break;
                }
                switch(ttsRequest)
                {
                case TTSRequest_Start:
                        info("TTSProcess() - Got TTSRequest_Start");
                        ProcessTTSRequest_Start();
                        break;
                case TTSRequest_Next:
                        info("TTSProcess() - Got TTSRequest_Next");
                        ProcessTTSRequest_Next();
                        break;
                case TTSRequest_Stop:
                        info("TTSProcess() - Got TTSRequest_Stop");
                        ProcessTTSRequest_Stop();
                        break;
                case TTSRequest_Exit:
                        info("TTSProcess() - Got TTSRequest_Exit");
                        done = true;
                        break;
                default:
                        error("TTSProcess() unknown request %d", request);
                        break;
                }
                lastRequest = ttsRequest;
        }
#ifdef  AVCORE
        deallocate_SSB(m_SSBPtr);
#else
        delete m_SSBPtr;
#endif
        info("End Thread TTSProcess()");
}
void
AcuvoiceTTSChannel::ProcessTTSRequest_Start(void)
{
        if (startedSynth)
        {
                warning("TTS threads out of sync");
                ::close_SSB(m_SSBPtr);
        }
```

```
        //Initialize SSB structure to synthesize a string.
        int rc = ::initialize_SSB( m_SSBPtr, m_pString, ACU_STRING);
        if(rc != 0)
        {
                error("Cannot Initialize SSB Variable (%d)", rc);
                dataAvail = 0;
                return;
        }
        dataAvail = ::isSeg_Available( m_SSBPtr );
        startedSynth = true;
        ProcessTTSRequest_Next();          // Get the first block
}
// synSize will be set to whatever data size we get.
//      dataAvail is also set.
void
AcuvoiceTTSChannel::ProcessTTSRequest_Next(void)
{
        int rc;
        if (!startedSynth)
        {
                warning("TTS threads out of sync (%d, %d)", startedSynth, synSize);
                dataAvail = 0;
        }
        if(dataAvail)
        {
                DWORD           pSynSize;
                // get ready the first buffer
#ifdef  AVCORE
                rc = ::synth_to_buffer(m_SSBPtr, (unsigned char *) audioBufInPtr, &pSynSize);
#else
                rc = ::synth_to_buffer(m_SSBPtr, audioBufInPtr, &pSynSize);
#endif
                if(rc != 0)
                {
                        diag("synth_to_buffer failed (%d)", rc);
                        dataAvail = 0;
                }
                else
                {
                        int tempDataAvail = ::isSeg_Available( m_SSBPtr );
                        diag("TTS: Next buffer of %d is ready", pSynSize);
                        Synchronized sync(m_WorkerMonitor);
                        synSize = pSynSize;
                        dataAvail = tempDataAvail;
                }
        }
}
void
AcuvoiceTTSChannel::ProcessTTSRequest_Stop(void)
{
        diag("TTS StopTransmit()");
        if (startedSynth)
                ::close_SSB(m_SSBPtr);
        startedSynth = false;
        dataAvail = 0;
        if(m_pString)
```

```
        {
                delete [] m_pString;
        }
        m_pString = NULL;
        {
        Synchronized sync(m_StateMonitor);
        m_State = TTS_Idle;
        sync.notify();
        }
        TTSChannel::NotifySettled();
}
```

Document Page: 1

```
/*
 * AcuvoiceTTSSP.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:34:26 $
 */
static const char* id="$Id: AcuvoiceTTSSP.cpp,v 1.5 2000/12/07 03:34:26 Mitsuru Oshima Exp $";
#include <stdio.h>
#include <windows.h>
#include "AcuvoiceTTSChannel.h"
#include "AcuvoiceTTSSP.h"
#include "MTSLib/MTSServer.h"
#include "MTSLib/MediaServiceProvider.h"
const char const *AcuvoiceTTSSP::cLogType="ACUV_TTS";
AcuvoiceTTSSP::~AcuvoiceTTSSP(void)
{
}
// virtual
bool
AcuvoiceTTSSP::Setup(void)
{
        //
        // For each telephony channel, try to allocate a TTS channel
        //
        std::list<Channel *>::size_type numchn =
                        GetMTSServer()->GetMediaServiceProvider()->GetNumChannels();
        numchn >>= 1;
        for (int i = 0; i < (int)numchn; i++)
        {
                AcuvoiceTTSChannel *pNewChannel = new AcuvoiceTTSChannel(this, cLogType, i);
                AddChannel(pNewChannel);
        }
        return true;
}
// virtual
bool
AcuvoiceTTSSP::Teardown(void)
{
        bool ret = true;
        // Channels are deleted by superclass
        //
        std::list<Channel *>::iterator iter = cChannels.begin();
        while (iter != cChannels.end())
        {
                if (!((AcuvoiceTTSChannel *)(*iter))->Shutdown()) {
                        ret = false;
                        error("Shutdown failed for Acuvoice Channel %p.", *iter);
                }
                iter++;
        }
```

```
        return ret;
}
TTSServiceProvider *
CreateTTSServiceProvider(MTSServer *pServer)
{
        return new AcuvoiceTTSSP(pServer);
}
```

Document Page 250

```
/*
 * AddPhraseToDynamicGrammarMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:44 $
 */
static const char* id="$Id: AddPhraseToDynamicGrammarMsg.cpp,v 1.2 2000/12/07 03:21:44 Mitsuru Oshima Exp
$";
#include "AddPhraseToDynamicGrammarMsg.h"
// virtual
void
AddPhraseToDynamicGrammarMessage::Serialize(void)
{
        DynamicGrammarPhraseMessage::Serialize();
        InsertString(cPhraseText);
        InsertString(cPhraseNL);
        InsertUlong(cProbability);
}
// virtual
void
AddPhraseToDynamicGrammarMessage::Deserialize(void)
{
        DynamicGrammarPhraseMessage::Deserialize();
        ExtractString(cPhraseText);
        ExtractString(cPhraseNL);
        cProbability = ExtractUlong();
}
```

Document ID: 20020130-0000

```
/*
 * Address.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:21:44 $
 */
static const char* id="$Id: Address.cpp,v 1.4 2000/12/07 03:21:44 Mitsuru Oshima Exp $";
#include "Address.h"
#include "VPPMsg.h"
void
Address::Serialize(VPPMessage *pMsg)
{
        pMsg->InsertString(cAddress);
}
void
Address::Deserialize(VPPMessage *pMsg)
{
        pMsg->ExtractString(cAddress);
}
```

```
/*
 * AppendCompositePromptDTMFMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:44 $
 */
static const char* id="$Id: AppendCompositePromptDTMFMsg.cpp,v 1.2 2000/12/07 03:21:44 Mitsuru Oshima Exp
$";
#include "AppendCompositePromptDTMFMsg.h"
// virtual
void
AppendCompositePromptDTMFMessage::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        InsertString(cDTMFString);
}
// virtual
void
AppendCompositePromptDTMFMessage::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        ExtractString(cDTMFString);
}
```

```
/*
 * AppendCompositePromptSilenceMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:44 $
 */
static const char* id="$Id: AppendCompositePromptSilenceMsg.cpp,v 1.3 2000/12/07 03:21:44 Mitsuru Oshima Exp
$";
#include "AppendCompositePromptSilenceMsg.h"
// virtual
void
AppendCompositePromptSilenceMessage::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        InsertUlong(cDuration);
}
// virtual
void
AppendCompositePromptSilenceMessage::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        cDuration = ExtractUlong();
}
```

```
/*
 * CallConnectedNtfy.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:21:45 $
 */
static const char* id="$Id: CallConnectedNtfy.cpp,v 1.4 2000/12/07 03:21:45 Mitsuru Oshima Exp $";
#include "CallConnectedNtfy.h"
// virtual
void
CallConnectedNotification::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        cRequest.Serialize(this);
}
// virtual
void
CallConnectedNotification::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        cRequest.Deserialize(this);
}
```

```
/*
 * CallProgressNtfy.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.6 $
 * $Date: 2000/12/07 03:21:45 $
 */
static const char* id="$Id: CallProgressNtfy.cpp,v 1.6 2000/12/07 03:21:45 Mitsuru Oshima Exp $";
#include "CallProgressNtfy.h"
// virtual
void
CallProgressNotification::Serialize(void)
{
        RequestTokenMessageBase::Serialize();
        InsertUshort((unsigned short)cNotification);
}
// virtual
void
CallProgressNotification::Deserialize(void)
{
        RequestTokenMessageBase::Deserialize();
        cNotification = (CallProgressT)ExtractUshort();
}
```

```
/*
 * CdrLogClient.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:19:46 $
 */
static const char* id="$Id: CdrLogClient.cpp,v 1.4 2000/12/07 03:19:46 Mitsuru Oshima Exp $";
#include <afxwin.h>
// Wrapper around Event Log Client
#include "VPPMsg/Address.h"
#include "VPPMsg/CDRInfo.h"
#include "EventLogClient/EventLogClient.h"
#include "CdrLogClient.h"
const char CDRLogClient::AgentNotificationCall[] = "agent_notification";
const char CDRLogClient::InboundCall[] = "inbound";
const char CDRLogClient::OutboundCall[] = "outbound";
CDRLogClient::CDRLogClient()
        :m_pEventLogClient(NULL),
        m_pCdrInfo(NULL)
{
}
CDRLOG_STATUS CDRLogClient::StartCall(CDRInfo  *pCdrInfo,
                                                                Address *Target, Address
*CallingNumber)
{
        CDRLOG_STATUS status = CDRLOG_SUCCESS;
        if(pCdrInfo->GetServiceName().empty())
        {
                status = CDRLOG_INFO_SERVICE_ERROR;
        }
        else if(pCdrInfo->GetBillingAccount().empty())
        {
                status = CDRLOG_INFO_ACCT_ERROR;
        }
        else if(pCdrInfo->GetCallFlavor().empty())
        {
                status = CDRLOG_INFO_CALLTYPE_ERROR;
        }
        else
        {
                if(stricmp(pCdrInfo->GetCallFlavor().c_str(), CDRLogClient::AgentNotificationCall) == 0)
                {
                        m_pCdrInfo = pCdrInfo;
                        // create new event log session
                        m_pEventLogClient = new CEventLogClient(LOGSESSION_AGENT_NOTIFICATION,
                                                                        m_pCdrInfo-
>GetBillingAccount().c_str(),
                                                                        m_pCdrInfo-
>GetServiceName().c_str());
                        m_pEventLogClient->Notify(AGENT_NOTIFICATION_START_CALL,
```

```
                                                                    CallingNumber-
>GetString().c_str(),
                                                                    Target->GetString().c_str());
                        m_pEventLogClient->Notify(AGENT_NOTIFICATION_CONNECT);
                }
                else
                {
                        //we do not process other then agent_notification events for now
                        status = CDRLOG_INFO_CALLTYPE_ERROR;
                }
        }
        return status;
}
CDRLOG_STATUS CDRLogClient::EndCall()
{
        if(m_pEventLogClient)
        {
                m_pEventLogClient->Notify(AGENT_NOTIFICATION_HANGUP);
                delete m_pEventLogClient;
                m_pEventLogClient = NULL;
        }
        return CDRLOG_SUCCESS;
}
CDRLogClient::~CDRLogClient()
{
        if(m_pEventLogClient)
        {
                delete m_pEventLogClient;
        }
}
```

```
/*
 * CallTokenMgr.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:19:45 $
 */
static const char* id="$Id: CallTokenMgr.cpp,v 1.2 2000/12/07 03:19:45 Mitsuru Oshima Exp $";
#include "CallTokenMgr.h"
#include "VPPMsg/CallToken.h"
class ID_TO_CT_Table : public Hashtable<CORBA::LongLong, CallToken> {
public:
   ID_TO_CT_Table() : Hashtable<CORBA::LongLong, CallToken>() {}
   virtual ~ID_TO_CT_Table() {
      clear();
   }
   int _hash(CORBA::LongLong value) {
      return (int)value;
   }
   CORBA::LongLong _copy(CORBA::LongLong key) {
      return key;
   }
   bool _keyEquals(CORBA::LongLong k1, CORBA::LongLong k2) {
      return k1 == k2;
   }
   bool _valueEquals(CallToken v1, CallToken v2) {
      return v1 == v2;
   }
   void _deleteKey(CORBA::LongLong key) { // no output
   }
};
CORBA::LongLong
Hashtable<CORBA::LongLong, CallToken>::NULL_KEY = 0;
CallToken
Hashtable<CORBA::LongLong, CallToken>::NULL_VALUE;
class CT_TO_ID_Table : public Hashtable<CallToken, CORBA::LongLong> {
public:
   CT_TO_ID_Table() : Hashtable<CallToken,CORBA::LongLong>() {}
   virtual ~CT_TO_ID_Table() {
      clear();
   }
   int _hash(CallToken ct) {
      return (int)ct.GetSequenceNumber();
   }
   CallToken _copy(CallToken ct) {
      return ct;
   }
   bool _keyEquals(CallToken k1, CallToken k2) {
      return k1 == k2;
   }
   bool _valueEquals(CORBA::LongLong v1, CORBA::LongLong v2) {
```

```
    return v1 == v2;
  }
  void _deleteKey(CallToken ct) { // no output
  }
};
CallToken
Hashtable<CallToken,CORBA::LongLong>::NULL_KEY;
CORBA::LongLong
Hashtable<CallToken,CORBA::LongLong>::NULL_VALUE = 0;
Hashtable<CORBA::LongLong, CallToken>*
CallTokenMgr::id_to_ct = new ID_TO_CT_Table();
Hashtable<CallToken,CORBA::LongLong>*
CallTokenMgr::ct_to_id = new CT_TO_ID_Table();
Queue<CORBA::LongLong>
CallTokenMgr::tokens;
void
CallTokenMgr::Register(CORBA::LongLong id,
                       CallToken& token)
{
  Synchronized sync(tokens);
  id_to_ct->put(id, token);
  ct_to_id->put(token, id);
  tokens.push(id);
  // TBD. change this!
  if (tokens.size() > 100) {
    CORBA::LongLong id = tokens.pop();
    CallToken ct = id_to_ct->get(id);
    id_to_ct->remove(id);
    ct_to_id->remove(ct);
  }
}
CallToken
CallTokenMgr::GetCallToken(CORBA::LongLong id)
{
  Synchronized sync(tokens);
  return id_to_ct->get(id);
}
CORBA::LongLong
CallTokenMgr::GetCallID(CallToken& token)
{
  Synchronized sync(tokens);
  return ct_to_id->get(token);
}
```

```
/*
 * CDRInfo.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:21:45 $
 */
static const char* id="$Id: CDRInfo.cpp,v 1.4 2000/12/07 03:21:45 Mitsuru Oshima Exp $";
#include "CDRInfo.h"
#include "VPPMsg.h"
CDRInfo&
CDRInfo::operator=(const CDRInfo& other)
{
        if(!other.cServiceName.empty())
        {
                cServiceName = other.cServiceName.c_str();
        }
        if(!other.cBillingAccount.empty())
        {
                cBillingAccount = other.cBillingAccount.c_str();
        }
        if(!other.cCallFlavor.empty())
        {
                cCallFlavor = other.cCallFlavor.c_str();
        }
        return *this;
}
void
CDRInfo::Serialize(VPPMessage *pMsg)
{
        pMsg->InsertString(cServiceName);
        pMsg->InsertString(cBillingAccount);
        pMsg->InsertString(cCallFlavor);
}
void
CDRInfo::Deserialize(VPPMessage *pMsg)
{
        pMsg->ExtractString(cServiceName);
        pMsg->ExtractString(cBillingAccount);
        pMsg->ExtractString(cCallFlavor);
}
```

```
/*
 * CDRInfoNtfy.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:45 $
 */
static const char* id="$Id: CDRInfoNtfy.cpp,v 1.2 2000/12/07 03:21:45 Mitsuru Oshima Exp $";
#include "CDRInfoNtfy.h"
// virtual
void
CDRInfoNotification::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        cCDRInfo.Serialize(this);
}
// virtual
void
CDRInfoNotification::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        cCDRInfo.Deserialize(this);
}
```

```
/*
 * CallToken.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.8 $
 * $Date: 2000/12/07 03:21:46 $
 */
static const char* id="$Id: CallToken.cpp,v 1.8 2000/12/07 03:21:46 Mitsuru Oshima Exp $";
#include "CallToken.h"
#include "VPPMsg.h"
void
CallToken::Serialize(VPPMessage *pMsg)
{
        pMsg->InsertUshort(cSpan);
        pMsg->InsertUshort(cChannel);
        pMsg->InsertUlong(cSequenceNumber);
        pMsg->InsertUlong(cTSPPrivate);
#if 0
        cMTSVirtualMachineID.Serialize(pMsg);
#endif
}
void
CallToken::Deserialize(VPPMessage *pMsg)
{
        cSpan = pMsg->ExtractUshort();
        cChannel = pMsg->ExtractUshort();
        cSequenceNumber = pMsg->ExtractUlong();
        cTSPPrivate = pMsg->ExtractUlong();
#if 0
        cMTSVirtualMachineID.Deserialize(pMsg);
#endif
}
```

```
/*
 * Channel.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.22 $
 * $Date: 2000/12/07 03:19:46 $
 */
static const char* id="$Id: Channel.cpp,v 1.22 2000/12/07 03:19:46 Mitsuru Oshima Exp $";
#include "Channel.h"
#include "VPPMsg/FailureNtfy.h"
#include "TelephonyChannel.h"
#include "VPPMsg/Property.h"
#include "SessionNode.h"
static char *ChannelStrings[4] = {"Telephony", "Media", "Speech", "TTS"};
static char *StateStrings[3] = {"S_Idle", "S_InSession", "S_Settling"};
// virtual
Channel::~Channel(void)
{
}
bool
Channel::SetState(Channel::ChannelState state)
{
   Synchronize sync(cStateLock);
   if ((state == Channel::S_Idle) || ((state - cState) == 1))
   {
      cState = state;
      return true;
   }
   else
   {
      cState = state;  // may want to remove this later
      error("bad state transitioning from state %d to state %d.",
          cState, state);
      return false;
   }
}
// virtual
void
Channel::NotifyBoundToSession(SessionNode *pSessionNode)
{
   if (pSessionNode)
   {
      SetState(S_InSession);
      cSession = pSessionNode;
      setLogID("<%d>[%ld]",cID, cSession->GetID());
      diag("bound to session : state = %s",
          StateStrings[GetState()]);
   }
   else
   {
      if (cRequiresSettlement)
```

```
        SetState(S_Settling);
      else
      {
        SetState(S_Idle);
        cSession = NULL;
      }
      setLogID("<%d>[unbound]",cID);
      diag("unbound from session : state = %s",
          StateStrings[GetState()]);
    }
    if (GetState() != S_InSession)
    {
      cCurrentCallToken.Invalidate();
    }
}
// virtual
void
Channel::NotifyBoundToCall(CallToken& callToken)
{
        cCurrentCallToken = callToken;
    setLogID("<%d>[%ld]",cID, cSession->GetID());
}
// virtual
void
Channel::NotifyPropertySet(Property& prop)
{
}
// virtual
bool
Channel::NotifyCallAboutToBeDropped(SessionNode* session, CallToken& callToken)
{
        return true;
}
void
Channel::NotifySettled(void)
{
    if (GetState() == S_Settling)
    {
      SetState(S_Idle);
      cSession = NULL;
      debug("NotifySettled : state = %s",
          StateStrings[GetState()]);
    }
    else
    {
      warning("NotifySettled : state = %s",
          StateStrings[GetState()]);
    }
}
bool
Channel::VerifySession(SessionNode *pSession)
{
        return (pSession && (pSession == cSession) && !pSession->IsGoingAway());
}
bool
Channel::Shutdown(void)
```

```
{
        return true;
}
```

Document not found

```
/*
 * CmdLine.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:22:14 $
 */
static const char* id="$Id: CmdLine.cpp,v 1.3 2000/12/07 03:22:14 Mitsuru Oshima Exp $";
#include "CmdLine.h"
void
CommandLine::KillNodes(void)
{
        std::list<CommandLineNode *>::iterator iter = cNodes.begin();

        while (iter != cNodes.end())
        {
                CommandLineNode *pNode = *iter;
                delete pNode;
                iter++;
        }
}
void
CommandLine::Setup(int ac, char **av)
{
        KillNodes();
        for (int i = 1; i < ac; i++)
        {
                CommandLineNode *pNode = new CommandLineNode;

                char *ptr = av[i];
                char *endptr = ptr;

                while (*endptr)
                {
                        endptr++;
                        if (*endptr == '=')
                        {
                                break;
                        }
                }
                pNode->cAttr.assign(ptr, endptr - ptr);
                if (*endptr)
                        endptr++;           // Past '='
                if (*endptr)
                        pNode->cValue = endptr;
                cNodes.push_back(pNode);
        }
}
const gm::string&
CommandLine::GetValue(const gm::string& attr)
{
```

```
	std::list<CommandLineNode *>::iterator iter = cNodes.begin();

	while (iter != cNodes.end())
	{
		if ((*iter)->cAttr == attr)
			break;
		iter++;
	}
	return iter != cNodes.end() ? (*iter)->cValue : gm::string::Nullref();
}
bool
CommandLine::DoesAttrAppear(const gm::string& attr)
{
	std::list<CommandLineNode *>::iterator iter = cNodes.begin();

	while (iter != cNodes.end())
	{
		if ((*iter)->cAttr == attr)
			break;
		iter++;
	}
	return iter != cNodes.end();
}
```

Document Page

```
/*
 * CMSClient.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:20:45 $
 */
static const char* id="$Id: CMSClient.cpp,v 1.5 2000/12/07 03:20:45 Mitsuru Oshima Exp $";
#include <windows.h>
#include <stdio.h>
#include "CMSClient.h"
#include "VPPUtils/VPPDebug.h"
CMSClient::CMSClient(IPCEndpoint *pIPCEndpoint)
        : CMSEndpoint(pIPCEndpoint)
{
        SetIsClient(true);
        cRetryInterval = DefaultRetryInterval;
        cDestroyingEvent = (unsigned long)::CreateEvent(NULL, TRUE, FALSE, NULL);
}
// virtual
CMSClient::~CMSClient(void)
{
        ::SetEvent((HANDLE)cDestroyingEvent);
}
bool
CMSClient::Connect(unsigned short NumUserObjects, unsigned long *UserObjects)
{
        bool ret;
        bool aborted = false;
        HANDLE *pHandles = new HANDLE[NumUserObjects + 1];
        pHandles[0] = (HANDLE)cDestroyingEvent;
        for (int i = 0; i < NumUserObjects; i++)
                pHandles[i + 1] = (HANDLE)UserObjects[i];
        do
        {
                DWORD dwRet = ::WaitForMultipleObjects(NumUserObjects + 1, pHandles, FALSE, 0);
                if (dwRet >= WAIT_OBJECT_0 && dwRet <= (WAIT_OBJECT_0 + NumUserObjects))
                        aborted = true;
                else
                {
                        debug::printf("CMSClient", "CMSClient %p attempting connection.\n", this);
                        ret = cEndpoint->Connect();
                        if (ret)
                                NotifyPeerConnected();
                        else if (cRetryInterval)
                                ::Sleep(cRetryInterval);
                }
        } while (!ret && !aborted && cRetryInterval);
        delete [] pHandles;
        return ret;
}
```

```
bool
CMSClient::Disconnect(void)
{
	bool ret = cEndpoint->Disconnect();
	NotifyPeerDisconnected(false);
	return ret;
}
// virtual
bool
CMSClient::NotifyPeerDisconnected(bool FarEnd)
{
	bool ret = CMSEndpoint::NotifyPeerDisconnected(FarEnd);
	if (FarEnd && cRetryInterval)
		return Connect();
	else
		return ret;
}
```

Document #: 1-000000-00

```
/*
 * CMSServerNode.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:20:47 $
 */
static const char* id="$Id: CMSServerNode.cpp,v 1.4 2000/12/07 03:20:47 Mitsuru Oshima Exp $";
#include "CMSServer.h"
#include "CMSServerNode.h"
// virtual
CMSServerNode::~CMSServerNode(void)
{
}
// virtual
MESSAGEHANDLER
CMSServerNode::FindMessageHandler(unsigned short cmdid, void **userdata)
{
        MESSAGEHANDLER handler = cServer->FindMessageHandler(cmdid, userdata);
        if (!handler)
                handler = CMSEndpoint::FindMessageHandler(cmdid, userdata);
        return handler;
}
// virtual
bool
CMSServerNode::NotifyPeerDisconnected(bool FarEnd)
{
        bool ret = CMSEndpoint::NotifyPeerDisconnected(FarEnd);
        ret &= cServer->NotifyServerNodeDisconnected(this);
        return ret;
}
```

```
/*
 * CMSEventArray.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:20:45 $
 */
static const char* id="$Id: CMSEventArray.cpp,v 1.4 2000/12/07 03:20:45 Mitsuru Oshima Exp $";
#define _WIN32_WINNT 0x0400
#include <windows.h>
#include "CMSEventArray.h"
CMSEventArray::CMSEventArray(void)
{
        cNumEvents = 2;
        cEvents = new unsigned long [2];
        cEvents[0] = (unsigned long)::CreateEvent(NULL, TRUE, FALSE, NULL);
        cEvents[1] = (unsigned long)::CreateEvent(NULL, TRUE, FALSE, NULL);
        cDispatchOKEvent = (unsigned long)::CreateEvent(NULL, TRUE, TRUE, NULL);
        cPingSendTimerEvent = (unsigned long)::CreateWaitableTimer(NULL, TRUE, NULL);
        cPingRecvTimerEvent = (unsigned long)::CreateWaitableTimer(NULL, TRUE, NULL);
}
CMSEventArray::~CMSEventArray(void)
{
        ::CloseHandle((HANDLE)cEvents[0]);
        ::CloseHandle((HANDLE)cEvents[1]);
        ::CloseHandle((HANDLE)cDispatchOKEvent);
        ::CloseHandle((HANDLE)cPingSendTimerEvent);
        ::CloseHandle((HANDLE)cPingRecvTimerEvent);
        delete [] cEvents;
}
void
CMSEventArray::Reset(void)
{
        ::ResetEvent((HANDLE)cEvents[0]);
        ::ResetEvent((HANDLE)cEvents[1]);
        ::SetEvent((HANDLE)cDispatchOKEvent);
}
```

```
/*
 * CMSMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.10 $
 * $Date: 2000/12/07 03:20:46 $
 */
static const char* id="$Id: CMSMsg.cpp,v 1.10 2000/12/07 03:20:46 Mitsuru Oshima Exp $";
#include <time.h>
#include "CMSMsg.h"
#include "IPCEndpoint.h"
#include "VPPUtils/Synchronize.h"
#include "VPPUtils/VPPDebug.h"
CMSMessage::SerialNumberT CMSMessage::cSerial = 1;
CriticalSection CMSMessage::cCriticalSection;
#define CHUNKSIZE 64
// virtual
CMSMessage::~CMSMessage(void)
{
        DeleteData();
}
CMSMessage&
CMSMessage::operator=(const CMSMessage& other)
{
        cFlags = other.cFlags;
        cSerialNo = other.cSerialNo;
        cReplyToSerialNo = other.cReplyToSerialNo;
        cCommandID = other.cCommandID;
        cUserdata = other.cUserdata;
        cDataLength = other.cDataLength;
        cAllocated = other.cAllocated;
        if (other.cAllocated)
        {
                cData = new char [other.cAllocated];
                debug::printf("CMSMessageAlloc", "Allocated %d bytes at %p\n", other.cAllocated,
                        cData);
                cParse = cData + (other.cParse - other.cData);
                memcpy(cData, other.cData, other.cDataLength);
        }
        else
                cData = cParse = 0;

        return *this;
}
void
CMSMessage::DeleteData(void)
{
        delete [] cData;
        cData = 0;
        cDataLength = 0;
        cAllocated = 0;
```

```
}
void
CMSMessage::CheckExpansion(unsigned long numbytes)
{
        if (cAllocated < (cDataLength + numbytes))
        {
                unsigned long size = ((((cDataLength + numbytes) + CHUNKSIZE - 1)
                        / CHUNKSIZE) * CHUNKSIZE);
                char *newdata = new char [size];
                if (cData)
                        memcpy(newdata, cData, cDataLength);
                cParse = newdata + (cParse - cData);
                delete [] cData;
                debug::printf("CMSMessageAlloc", "Allocated %d bytes at %p\n", size,
                        newdata);
                cData = newdata;
                cAllocated = size;
        }
}
void
CMSMessage::ReadFromPtr(char **ptr)
{
        unsigned long len = 0;
        cFlags = ExtractUshort(ptr);
        cSerialNo = (SerialNumberT)ExtractUshort(ptr);
        cReplyToSerialNo = (SerialNumberT)ExtractUshort(ptr);
        cCommandID = ExtractUshort(ptr);
        cUserdata = ExtractUlong(ptr);
        len = ExtractUlong(ptr);
        cDataLength = 0;

        unsigned long BytesRead = 0;
        delete [] cData;
        cData = 0;
        if (len)
        {
                CheckExpansion(len);
                cDataLength = len;
                ExtractRaw((char *)cData, cDataLength, ptr);
        }
        cParse = cData;
}
bool
CMSMessage::ReadFrom(IPCEndpoint *pEndpoint, unsigned short NumUserObjects,
        unsigned long *UserObjects)
{
        bool ret = true;
        unsigned long len = 0;
        debug::printf("CMSMessage", "Beginning message(%p)  read\n",
                this, cSerialNo, cCommandID);
        pEndpoint->BeginAtomicRead();
        if (!pEndpoint->Read(cFlags, NumUserObjects, UserObjects) ||
                !pEndpoint->Read(cSerialNo, NumUserObjects, UserObjects) ||
                !pEndpoint->Read(cReplyToSerialNo, NumUserObjects, UserObjects) ||
                !pEndpoint->Read(cCommandID, NumUserObjects, UserObjects) ||
                !pEndpoint->Read(cUserdata, NumUserObjects, UserObjects) ||
```

```
                !pEndpoint->Read(len, NumUserObjects, UserObjects))
                ret = false;
        cDataLength = 0;
        unsigned long BytesRead = 0;
        delete [] cData;
        cData = 0;
        if (ret && len)
        {
                CheckExpansion(len);
                cDataLength = len;

                ret = pEndpoint->Read((unsigned char *)cData, cDataLength, &BytesRead,
                        NumUserObjects, UserObjects);

                if (BytesRead != cDataLength)
                        ret = false;
        }
        cParse = cData;
        pEndpoint->EndAtomicRead();
        debug::printf("CMSMessage", "Done with read(%p) of S/N %d, CmdID %d\n",
                this, cSerialNo, cCommandID);
        if (cCommandID == (unsigned short)324)
                debug::printf("DelayDebugging", "Done with read(%p) of
M_CMD_CREATECOMPOSITEPROMPT S/N %d, CmdID %d\n",
                this, cSerialNo, cCommandID);
        return ret;
}
bool
CMSMessage::WriteTo(IPCEndpoint *pEndpoint, unsigned short NumUserObjects,
        unsigned long *UserObjects)
{
        bool ret = true;

        AssignSerialNo();
        if (cCommandID != C_NTFY_PING)
                debug::printf("CMSMessage", "Beginning write of S/N %d, CmdID %d\n",
                        cSerialNo, cCommandID);
        pEndpoint->BeginAtomicWrite();

        SetValid();
        Serialize();

        if (!pEndpoint->Write(cFlags, NumUserObjects, UserObjects) ||
                !pEndpoint->Write(cSerialNo, NumUserObjects, UserObjects) ||
                !pEndpoint->Write(cReplyToSerialNo, NumUserObjects, UserObjects) ||
                !pEndpoint->Write(cCommandID, NumUserObjects, UserObjects) ||
                !pEndpoint->Write(cUserdata, NumUserObjects, UserObjects) ||
                !pEndpoint->Write(cDataLength, NumUserObjects, UserObjects))
                ret = false;
        if (ret)
        {
                unsigned long BytesWritten = 0;
                if (cDataLength)
                {
                        ret = pEndpoint->Write((unsigned char *)cData, cDataLength,
                                &BytesWritten, NumUserObjects, UserObjects);
```

```
                        if (BytesWritten != cDataLength)
                                ret = false;
                }
        }
        pEndpoint->EndAtomicWrite();
        if (cCommandID != C_NTFY_PING)
                debug::printf("CMSMessage", "Done with write of S/N %d, CmdID %d\n",
                        cSerialNo, cCommandID);
        return ret;
}
void
CMSMessage::InsertUchar(unsigned char uc, char **ptr)
{
        if (!ptr)
                ptr = &cParse;

        CheckExpansion(1);
        *(*ptr) = uc;
        (*ptr)++;
        cDataLength++;
}
void
CMSMessage::InsertUshort(unsigned short us, char **ptr)
{
        InsertUchar(((us >> 8) & 0xff), ptr);
        InsertUchar((us & 0xff), ptr);
}
void
CMSMessage::InsertUlong(unsigned long ul, char **ptr)
{
        InsertUshort((unsigned short)((ul >> 16) & 0xffff), ptr);
        InsertUshort((unsigned short)(ul & 0xffff), ptr);
}
void
CMSMessage::InsertPtr(void *ptr, char **parse)
{
        InsertUlong((unsigned long)ptr, parse);
}
void
CMSMessage::InsertString(const char *str, char **ptr)
{
        if (str)
                InsertRaw(str, strlen(str) + 1, ptr);
        else
                InsertRaw("", 1, ptr);
}
void
CMSMessage::InsertMessage(CMSMessage *pMessage, char **ptr)
{
        pMessage->Serialize();
        InsertUshort(pMessage->cFlags);
        InsertUshort(pMessage->cSerialNo);
        InsertUshort(pMessage->cReplyToSerialNo);
        InsertUshort(pMessage->cCommandID);
        InsertUlong(pMessage->cUserdata);
        InsertUlong(pMessage->cDataLength);
```

```
	InsertRaw(pMessage->GetData(), pMessage->GetDataLength(), ptr);
}
void
CMSMessage::InsertRaw(const char *buf, unsigned long len, char ** ptr)
{
	for (unsigned long i = 0; i < len; i++)
		InsertUchar((unsigned char)buf[i], ptr);
}
unsigned char
CMSMessage::ExtractUchar(char **ptr)
{
	if (!ptr)
		ptr = &cParse;

	unsigned char ret = (unsigned char)*(*ptr);
	(*ptr)++;
	return ret;
}
unsigned short
CMSMessage::ExtractUshort(char **ptr)
{
	return (unsigned short)((ExtractUchar(ptr) << 8) | ExtractUchar(ptr));
}
unsigned long
CMSMessage::ExtractUlong(char **ptr)
{
	return (unsigned long)((ExtractUshort(ptr) << 16) | ExtractUshort(ptr));
}
void *
CMSMessage::ExtractPtr(char **ptr)
{
	return (void *)ExtractUlong(ptr);
}
void
CMSMessage::ExtractString(char *str, unsigned long maxlen, char **ptr)
{
	unsigned long len = 0;
	unsigned char ch;
	do
	{
		ch = ExtractUchar(ptr);
		if (len < maxlen)
		{
			str[len] = (char)ch;
			len++;
		}
	} while (ch);
	str[maxlen] = 0;
}
void
CMSMessage::ExtractString(gm::string& str, char **ptr)
{
	unsigned char ch;
	str.erase();

	do
```

```
        {
                ch = ExtractUchar(ptr);
                if (ch)
                        str += (char)ch;
        } while (ch);
}
void
CMSMessage::ExtractMessage(CMSMessage *pMessage, char **ptr)
{
        pMessage->ReadFromPtr(ptr);
}
void
CMSMessage::ExtractRaw(char *buf, unsigned long len, char **ptr)
{
        while (len--)
                *buf++ = (char)ExtractUchar(ptr);
}
// virtual
void
CMSMessage::Serialize(void)
{
}
// virtual
void
CMSMessage::Deserialize(void)
{
}
void
CMSMessage::AssignSerialNo(void)
{
        Synchronize sync(cCriticalSection);
        cSerialNo = cSerial++;
        if (!cSerial)
                cSerial++;
}
```

```
/*
 * CMSMsgMap.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:20:46 $
 */
static const char* id="$Id: CMSMsgMap.cpp,v 1.3 2000/12/07 03:20:46 Mitsuru Oshima Exp $";
#include <memory.h>
#include "CMSMsgMap.h"
#include "VPPUtils/Synchronize.h"
CMSMessageMap::~CMSMessageMap(void)
{
        std::list<MessageMapEntry *>::iterator iter = cMap.begin();
        while (iter != cMap.end())
        {
                delete *iter;
                iter++;
        }
}
CMSMessageMap::MessageMapEntry *
CMSMessageMap::FindEntry(unsigned short cmdid)
{
        Synchronize sync(cCriticalSection);
        std::list<MessageMapEntry *>::iterator iter = cMap.begin();
        while (iter != cMap.end())
        {
                if ((*iter)->cCommandID == cmdid)
                        return *iter;
                iter++;
        }
        return 0;
}
bool
CMSMessageMap::AddMessageHandler(unsigned short cmdid, MESSAGEHANDLER handler,
        void *userdata)
{
        Synchronize sync(cCriticalSection);
        MessageMapEntry *pEntry;
        if ((pEntry = FindEntry(cmdid)))
        {
                pEntry->cHandler = handler;
                pEntry->cUserdata = userdata;
                return true;
        }
        MessageMapEntry *pNewEntry = new MessageMapEntry;
        pNewEntry->cCommandID = cmdid;
        pNewEntry->cHandler = handler;
        pNewEntry->cUserdata = userdata;
        cMap.push_back(pNewEntry);
        return true;
```

```
}
MESSAGEHANDLER
CMSMessageMap::FindMessageHandler(unsigned short cmdid, void **userdata)
{
	Synchronize sync(cCriticalSection);
	MessageMapEntry *pEntry = FindEntry(cmdid);
	if (pEntry)
	{
		if (userdata)
			*userdata = pEntry->cUserdata;
		return pEntry->cHandler;
	}
	return 0;
}
```

```
/*
 * CMSMsgQueue.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.7 $
 * $Date: 2000/12/07 03:20:46 $
 */
static const char* id="$Id: CMSMsgQueue.cpp,v 1.7 2000/12/07 03:20:46 Mitsuru Oshima Exp $";
#include <windows.h>
#include "CMSMsg.h"
#include "CMSMsgQueue.h"
#include "VPPUtils/Synchronize.h"
#include "VPPUtils/VPPDebug.h"
CMSMessageQueue::CMSMessageQueue(void)
{
        cMsgInQueueEvent = (unsigned long)::CreateEvent(NULL, TRUE, FALSE, NULL);
}
CMSMessageQueue::~CMSMessageQueue(void)
{
        std::list<CMSMessage *>::iterator iter = cMsgs.begin();

        while (iter != cMsgs.end())
        {
                delete *iter;
                iter++;
        }

        ::CloseHandle((HANDLE)cMsgInQueueEvent);
}
bool
CMSMessageQueue::InsertMessage(CMSMessage *pMsg)
{
        Synchronize sync(cCriticalSection);
        cMsgs.push_back(pMsg);
        if (cMsgs.size())
                ::SetEvent((HANDLE)cMsgInQueueEvent);
        return true;
}
void
CMSMessageQueue::RemoveMsg(CMSMessage *pMsg, bool DeleteMsg)
{
        Synchronize sync(cCriticalSection);
        cMsgs.remove(pMsg);
        if (!cMsgs.size())
                ::ResetEvent((HANDLE)cMsgInQueueEvent);
        debug::printf("CMSMsgQueue", "%ld : MessageQueue::Remove (%p)\n", cMsgs.size(), this);
        if (DeleteMsg)
                delete pMsg;
}
bool
CMSMessageQueue::IsMessageWaiting(CMSMessage **ppMsg, bool Remove)
```

```
{
        Synchronize sync(cCriticalSection);
        if (!cMsgs.size())
                return false;
        if (ppMsg)
                *ppMsg = cMsgs.back();
        if (Remove)
                RemoveMsg(cMsgs.back(), false);
        return true;
}
CMSMessage *
CMSMessageQueue::RemoveMessage(unsigned short NumUserEvents, unsigned long *UserEvents)
{
        CMSMessage *ret = 0;
        HANDLE *handles = new HANDLE [NumUserEvents + 1];
        handles[0] = (HANDLE)cMsgInQueueEvent;

        for (unsigned short i = 1; i <= NumUserEvents; i++)
                handles[i] = (HANDLE)UserEvents[i - 1];

        DWORD result = ::WaitForMultipleObjects(NumUserEvents + 1, handles,
                false, INFINITE);
        delete [] handles;
        if (result == WAIT_OBJECT_0)
        {
                // Msg is in the queue, so get it out
                //
                Synchronize sync(cCriticalSection);
                ret = cMsgs.front();
                RemoveMsg(ret, false);
        }
        return ret;
}
void
CMSMessageQueue::Clear(void)
{
        cMsgs.clear();
}
```

```
/*
 * CMSMultiClient.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:20:46 $
 */
static const char* id="$Id: CMSMultiClient.cpp,v 1.4 2000/12/07 03:20:46 Mitsuru Oshima Exp $";
#include "CMSMultiClient.h"
#include "VPPUtils/Synchronize.h"
CMSMultiClient::CMSMultiClient(void)
{
        ResetIterator();
}
// virtual
CMSMultiClient::~CMSMultiClient(void)
{
        std::list<CMSClient *>::iterator iter = cClients.begin();
        while (iter != cClients.end())
        {
                delete *iter;
                iter++;
        }
        cClients.clear();
}
void
CMSMultiClient::AddClient(CMSClient *pClient)
{
        Synchronize sync(cCriticalSection);
        cClients.push_back(pClient);
        ResetIterator();
}
void
CMSMultiClient::RemoveClient(CMSClient *pClient)
{
        Synchronize sync(cCriticalSection);
        cClients.remove(pClient);
        ResetIterator();
}
CMSClient *
CMSMultiClient::GetClient(unsigned short index)
{
        Synchronize sync(cCriticalSection);
        unsigned short i = 0;
        std::list<CMSClient *>::iterator iter = cClients.begin();
        while (iter != cClients.end() && i < index)
        {
                iter++;
                i++;
        }
        if (iter != cClients.end())
```

```
                return *iter;
        return 0;
}
CMSClient *
CMSMultiClient::GetNextClient(void)
{
        Synchronize sync(cCriticalSection);
        CMSClient *pRet = *cIter;
        IncrementIterator();
        return pRet;
}
void
CMSMultiClient::ResetIterator(void)
{
        Synchronize sync(cCriticalSection);
        cIter = cClients.begin();
}
void
CMSMultiClient::IncrementIterator(void)
{
        Synchronize sync(cCriticalSection);
        cIter++;
        if (cIter == cClients.end())
                ResetIterator();
}
unsigned short
CMSMultiClient::GetNumClients(void)
{
        Synchronize sync(cCriticalSection);
        return cClients.size();
}
CMSMessage::SerialNumberT
CMSMultiClient::BroadcastMessage(CMSMessage& msg)
{
        CMSMessage::SerialNumberT sn = (CMSMessage::SerialNumberT)0;
        Synchronize sync(cCriticalSection);
        std::list<CMSClient *>::iterator iter = cClients.begin();
        while (iter != cClients.end())
        {
                sn = (*iter)->WriteMessage(msg);
                iter++;
        }
        return sn;
}
```

```
/*
 * CMSServer.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.10 $
 * $Date: 2000/12/07 03:20:47 $
 */
static const char* id="$Id: CMSServer.cpp,v 1.10 2000/12/07 03:20:47 Mitsuru Oshima Exp $";
#include <windows.h>
#include <stdio.h>
#include "CMSServer.h"
#include "CMSServerNode.h"
#include "VPPUtils/Synchronize.h"
#include "VPPUtils/VPPDebug.h"
CMSServer::CMSServer(IPCEndpoint *pIPCEndpoint)
        : CMSEndpoint(pIPCEndpoint)
{
        // Servers have to run at high priority
        //
        ::SetPriorityClass(::GetCurrentProcess(), HIGH_PRIORITY_CLASS);
        //::SetPriorityClass(::GetCurrentProcess(), NORMAL_PRIORITY_CLASS);
}
// virtual
CMSServer::~CMSServer(void)
{
        std::list<CMSServerNode *>::iterator iter = cNodes.begin();
        while (iter != cNodes.end())
        {
                delete *iter;
                iter++;
        }
}
// virtual
bool
CMSServer::Listen(void)
{
        return cEndpoint->Listen();
}
bool
CMSServer::NotifyServerNodeDisconnected(CMSServerNode *pServerNode)
{
        Synchronize sync(cCriticalSection);
        cNodes.remove(pServerNode);
        delete pServerNode;
        return true;
}
bool
CMSServer::NewConnection(IPCEndpoint *pEndpoint)
{
        // Create a new CMSServerNode to service this new IPCEndpoint,
        // and add it to our list
```

```
	//
	Synchronize sync(cCriticalSection);
	CMSServerNode *pCMSServerNode = CreateServerNode(pEndpoint);
	cNodes.push_back(pCMSServerNode);
	if (pEndpoint->IsConnected())
		pCMSServerNode->NotifyPeerConnected();
	return true;
}
bool
CMSServer::ClientConnected(void)
{
	IPCEndpoint *pEndpoint = cEndpoint->ClientConnected();
	if (pEndpoint)
		return NewConnection(pEndpoint);
	return false;
}
int
CMSServer::WaitForClientConnect(unsigned short NumUserObjects, unsigned long *UserObjects)
{
	IPCEndpoint *pEndpoint = cEndpoint->WaitForClientConnect(NumUserObjects,
		UserObjects);

	if (pEndpoint)
	{
		if (NewConnection(pEndpoint))
			return 1;
	}
	else
	{
		DWORD result = ::WaitForMultipleObjects(NumUserObjects, (HANDLE *)UserObjects,
			false, 0);
		if (result >= WAIT_OBJECT_0 &&
			result <= (WAIT_OBJECT_0 + NumUserObjects - 1))
			return 2;
	}

	return 0;
}
// virtual
CMSServerNode *
CMSServer::CreateServerNode(IPCEndpoint *pIPCEndpoint)
{
	return new CMSServerNode(this, pIPCEndpoint);
}
unsigned short
CMSServer::GetNumOpenConnections(void)
{
	Synchronize sync(cCriticalSection);
	return cNodes.size();
}
CMSServerNode *
CMSServer::GetServerNode(unsigned short index)
{
	Synchronize sync(cCriticalSection);
	unsigned short i = 0;
	std::list<CMSServerNode *>::iterator iter = cNodes.begin();
```

```
        while (iter != cNodes.end() && i < index)
        {
                iter++;
                i++;
        }
        return (iter != cNodes.end()) ? *iter : 0;
}
CMSMessage::SerialNumberT
CMSServer::BroadcastMessage(CMSMessage& msg)
{
        CMSMessage::SerialNumberT sn = (CMSMessage::SerialNumberT)0;
        Synchronize sync(cCriticalSection);
        std::list<CMSServerNode *>::iterator iter = cNodes.begin();
        while (iter != cNodes.end())
        {
                sn = (*iter)->WriteMessage(msg);
                iter++;
        }
        return sn;
}
```

```
/*
 * CMSEndpoint.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.27 $
 * $Date: 2000/12/07 03:20:45 $
 */
static const char* id="$Id: CMSEndpoint.cpp,v 1.27 2000/12/07 03:20:45 Mitsuru Oshima Exp $";
#define _WIN32_WINNT 0x0400
#include <windows.h>
#include <stdio.h>
#include <process.h>
#include "CMSEndpoint.h"
#include "ipcpipe.h"
#include "VPPUtils/VPPDebug.h"
#include "VPPUTils/Synchronize.h"
#define DEFAULT_PING_PERIOD 500L
#define PING_SLACK_TIME 500L
#define MAX_PING_DECAY_LEVEL 2
CMSEndpoint::CMSEndpoint(IPCEndpoint *pIPCEndpoint)
{
        debug::printf("CMS", "Creating CMSEndpoint %p\n", this);
        cFlags = F_AUTODISPATCH;
        cReadThread = cDispatchThread = cSendPingThread = cRecvPingThread = 0;
        SetPingPeriod(0);
        SetIPCEndpoint(pIPCEndpoint);
        cPingDecayLevel = 0;
        debug::printf("CMS", "Done creating CMSEndpoint %p\n", this);
}
// virtual
CMSEndpoint::~CMSEndpoint(void)
{
        cEndpoint->Disconnect();
        ShutdownThreads();
        debug::printf("CMS", "Destroying CMSEndpoint %p\n", this);
        delete cEndpoint;
}
// virtual
bool
CMSEndpoint::NotifyPeerConnected(void)
{
        bool ret = true;
        unsigned thrdaddr;
        debug::printf("CMSEndpointConnection", "CMSEndpoint %p connected to peer.\n", this);
        cEvents.Reset();
        cMsgQueue.Clear();
        cReadThread = ::_beginthreadex(NULL, 0,  _ReadThreadFunc, this, 0,
                &thrdaddr);
        debug::printf("CMS", "Main thread is %ld\n", ::GetCurrentThreadId());
        if (!cReadThread)
                return false;
```

```
        // Clients send their preferred ping time here
        //
        if (IsClient())
        {
                CMSMessage msg(C_CMD_REQUESTPINGPERIOD);
                msg.SetUserdata(DEFAULT_PING_PERIOD);
                debug::printf("Ping", "Requesting ping period of %ld ms\n", DEFAULT_PING_PERIOD);
                WriteMessage(msg);
        }
        cSendPingThread = ::_beginthreadex(NULL, 0, _SendPingThreadFunc, this, 0,
                &thrdaddr);
        cRecvPingThread = ::_beginthreadex(NULL, 0, _RecvPingThreadFunc, this, 0,
                &thrdaddr);
        if (!cSendPingThread || !cRecvPingThread)
        {
                ::SetEvent((HANDLE)cEvents.GetTerminateThreadsEvent());
                return false;
        }
        if (IsAutoDispatch())
        {
                cDispatchThread = ::_beginthreadex(NULL, 0,          _DispatchThreadFunc,
                        this, 0, &thrdaddr);
                if (!cDispatchThread)
                {
                        ::SetEvent((HANDLE)cEvents.GetTerminateThreadsEvent());
                        return false;
                }
        }
        return ret;
}
// virtual
bool
CMSEndpoint::NotifyPeerDisconnected(bool FarEnd)
{
        debug::printf("CMSEndpointConnection", "CMSEndpoint %p disconnected from peer by the %s end.\n",
                this, FarEnd ? "far" : "near");
        ShutdownThreads();
        return true;
}
unsigned __stdcall
CMSEndpoint::_ReadThreadFunc(LPVOID ptr)
{
        debug::printf("MsgRead", "Read thread is %ld\n", ::GetCurrentThreadId());

        CMSEndpoint *pThis = (CMSEndpoint *)ptr;
        return pThis->ReadThreadFunc();
}
unsigned
CMSEndpoint::ReadThreadFunc(void)
{
        bool Disconnected = false;

        CMSMessage *pMsg;
        while (!Disconnected)
        {
                if ((pMsg = ReadMessage(1, cEvents.GetTerminateThreadsEventPtr())))
```

```
{
        ResetReceivePingTimer(true);

        if (pMsg->GetCommandID() < C_USER_CMDID_START)
        {
                // This is a CMS private message.  These messages get processed here, rather than
                // through the handler mechanism, for efficiency.
                //
                if (pMsg->GetCommandID() == C_NTFY_PING)
                {
                        debug::printf("Pings", "Received ping\n");
                }
                else if (pMsg->GetCommandID() == C_CMD_REQUESTPINGPERIOD)
                {
                        debug::printf("Ping",
                                "Client requested ping period of %ld ms\n", pMsg-
>GetUserdata());

                        CMSMessage msg(C_NTFY_SETPINGPERIOD);
//                      msg.SetUserdata(pMsg->GetUserdata());
                        msg.SetUserdata(0);
                        debug::printf("Ping", "Mandating ping period of %ld ms\n",
msg.GetUserdata());
                        WriteMessage(msg);
                        SetPingPeriod(msg.GetUserdata());
                }
                else if (pMsg->GetCommandID() == C_NTFY_SETPINGPERIOD)
                {
                        debug::printf("Ping",
                                "Server mandated ping period of %ld ms\n", pMsg-
>GetUserdata());
                        SetPingPeriod(pMsg->GetUserdata());
                }
        }
        else
        {
                unsigned short sn = pMsg->GetSerialNo();
                unsigned short cmd = pMsg->GetCommandID();
                debug::printf("MsgRead", "Read message S/N %d, CmdID %d\n",
                        sn, cmd);

                if (!cMsgQueue.InsertMessage(pMsg))
                {
                        debug::printf("MsgRead", "InsertMessage failed (%ld)!\n",
                                ::GetLastError());
                }
                else
                        debug::printf("MsgRead", "Queued message S/N %d, CmdId %d\n\n",
                                sn, cmd);
        }
}
else if (!cEndpoint->IsConnected())
        Disconnected = true;
else if (::WaitForSingleObject(
        (HANDLE)cEvents.GetTerminateThreadsEvent(), 0) == WAIT_OBJECT_0)
{
        debug::printf("MsgRead", "ReadThreadFunc got cEvent[1] signalled!\n");
```

```
                        break;
                }
        }
        ::CloseHandle((HANDLE)cReadThread);
        cReadThread = 0;
        if (Disconnected && !cEndpoint->NearEndDisconnected())
        {
                debug::printf("MsgRead", "ReadThreadFunc got FarEnd Disconnected\n");
                cEndpoint->Disconnect();
                NotifyPeerDisconnected(true);
        }
        return 0;
}
unsigned __stdcall
CMSEndpoint::_DispatchThreadFunc(LPVOID ptr)
{
        debug::printf("MsgDispatch", "Dispatch thread is %ld\n", ::GetCurrentThreadId());
        CMSEndpoint *pThis = (CMSEndpoint *)ptr;
        return pThis->DispatchThreadFunc();
}
unsigned
CMSEndpoint::DispatchThreadFunc(void)
{
        //::SetThreadPriority(::GetCurrentThread(), THREAD_PRIORITY_ABOVE_NORMAL);
        ::SetThreadPriority(::GetCurrentThread(), THREAD_PRIORITY_HIGHEST);
        debug::printf("MsgDispatch", "Waiting for message on thread %ld\n", ::GetCurrentThreadId());

        while (true)
        {
                // Wait for one of several things to happen:
                //
                // 1. A msg enters the queue (put there by ReadThreadFunc)
                // 2. The InTransactionEvent becomes signaled.  This means we
                //    are in a transaction and cannot autodispatch messages
                //    until we finish the transaction (InTransactionEvent is a
                //    per-thread event object)
                // 3. One of user-specified events is signaled.
                //
                CMSMessage *pMsg;
                debug::printf("MsgDispatch", "Waiting to dequeue next message\n");
                pMsg = cMsgQueue.RemoveMessage(cEvents.GetNumEvents(),
                        cEvents.GetEvents());
                if (pMsg)
                {
                        // Msg arived in queue
                        //
                        debug::printf("MsgDispatch", "Dequeued message S/N %d, CmdID %d\n",
                                pMsg->GetSerialNo(), pMsg->GetCommandID());
                        ProcessMessage(pMsg);
                        debug::printf("MsgDispatch", "Processed message S/N %d, CmdID %d\n",
                                pMsg->GetSerialNo(), pMsg->GetCommandID());
                        unsigned short sn = pMsg->GetSerialNo();
                        delete pMsg;
                        debug::printf("MsgDispatch", "Deleted message S/N %d\n", sn);
                }
                else
```

```
                {
                        if (::WaitForSingleObject((HANDLE)cEvents.GetInTransactionEvent(), 0) ==
                                WAIT_OBJECT_0)
                        {
                                // InTransactionEvent was signaled
                                //
                                debug::printf("MsgDispatch", "MsgWaitForMultipleObjects halted because of
cInTransactionEvent!\n");
                                // Wait for the DispatchOK event to be signaled
                                //
                                ::WaitForSingleObject((HANDLE)cEvents.GetDispatchOKEvent(), INFINITE);
                        }
                        else if (::WaitForSingleObject((HANDLE)cEvents.GetTerminateThreadsEvent(),
                                0) == WAIT_OBJECT_0)
                        {
                                // EndThread event signalled
                                //
                                debug::printf("MsgDispatch", "DispatchThreadFunc got TerminateThreadsEvent
signalled!\n");
                                break;
                        }
                }
        }
        ::CloseHandle((HANDLE)cDispatchThread);
        cDispatchThread = 0;
        return 0;
}
unsigned __stdcall
CMSEndpoint::_SendPingThreadFunc(LPVOID ptr)
{
        debug::printf("CMS", "Send Ping thread is %ld\n", ::GetCurrentThreadId());
        CMSEndpoint *pThis = (CMSEndpoint *)ptr;
        return pThis->SendPingThreadFunc();
}
unsigned
CMSEndpoint::SendPingThreadFunc(void)
{
        HANDLE handles[2];
        handles[0] = (HANDLE)cEvents.GetTerminateThreadsEvent();
        handles[1] = (HANDLE)cEvents.GetPingSendTimerEvent();
        CMSMessage pingmsg(C_NTFY_PING);
        ResetSendPingTimer();
        while (true)
        {
                // Wait for either the TerminateThreads or SendPingTimer events.
                //
                DWORD result = ::WaitForMultipleObjects(2, handles, FALSE, INFINITE);
                if (result == WAIT_OBJECT_0)
                {
                        debug::printf("CMS", "SendPingThreadFunc got TerminateThreadsEvent signalled!\n");
                        break;
                }
                else if (result == WAIT_OBJECT_0 + 1)
                {
                        // Send ping timer went off - send a ping message
                        //
```

```
                debug::printf("Pings", "Sending ping\n");
                WriteMessage(pingmsg);
            }
        }

        ::CloseHandle((HANDLE)cSendPingThread);
        cSendPingThread = 0;
        return 0;
}
unsigned __stdcall
CMSEndpoint::_RecvPingThreadFunc(LPVOID ptr)
{
        debug::printf("CMS", "Recv Ping thread is %ld\n", ::GetCurrentThreadId());
        CMSEndpoint *pThis = (CMSEndpoint *)ptr;
        return pThis->RecvPingThreadFunc();
}
unsigned
CMSEndpoint::RecvPingThreadFunc(void)
{
        HANDLE handles[2];
        handles[0] = (HANDLE)cEvents.GetTerminateThreadsEvent();
        handles[1] = (HANDLE)cEvents.GetPingReceiveTimerEvent();
        ResetReceivePingTimer(true);

        while (true)
        {
            // Wait for either the TerminateThreads or RecvPingTimer events.
            //
            DWORD result = ::WaitForMultipleObjects(2, handles, FALSE, INFINITE);
            if (result == WAIT_OBJECT_0)
            {
                debug::printf("CMS", "RcvPingThreadFunc got TerminateThreadsEvent signalled!\n");
                break;
            }
            else if (result == WAIT_OBJECT_0 + 1)
            {
                // Receive ping timer went off - this means we didn't get
                // a ping and we wanted one.  The other end of the pipe
                // is not pinging.
                //
                if (cPingDecayLevel == MAX_PING_DECAY_LEVEL)
                {
                    CMSMessage msg(C_NTFY_PINGFAILURE);
                    ProcessMessage(&msg);
                    debug::printf("Ping", "Ping Failure!\n");
                }
                else
                {
                    debug::printf("Ping", "Didn't get ping in time, increasing decay to %d\n",
                        cPingDecayLevel);
                    cPingDecayLevel++;
                }
                ResetReceivePingTimer(false);
            }
        }
        ::CloseHandle((HANDLE)cRecvPingThread);
```

```
        cRecvPingThread = 0;
        return 0;
}
bool
CMSEndpoint::MessagesWaiting(void)
{
        // You can't poll the messages in an AutoDispatch endpoint, unless
        // we're in a transaction
        //
        if (IsAutoDispatch() && ::WaitForSingleObject(
                (HANDLE)cEvents.GetInTransactionEvent(), 0) != WAIT_OBJECT_0)
                return false;

        debug::printf("CMS", "Checking for messages on thread %ld\n", ::GetCurrentThreadId());
        return cMsgQueue.IsMessageWaiting();
}
CMSMessage *
CMSEndpoint::GetNextMessage(void)
{
        // You can't poll the messages in an AutoDispatch endpoint, unless
        // we're in a transaction
        //
        if (IsAutoDispatch() && ::WaitForSingleObject(
                (HANDLE)cEvents.GetInTransactionEvent(), 0) != WAIT_OBJECT_0)
                return false;
        debug::printf("CMS", "Getting message from thread %ld...", ::GetCurrentThreadId());
        CMSMessage *pRet = 0;

        if ((pRet = cMsgQueue.RemoveMessage()))
        {
                debug::printf("CMS", "got it!\n");
        }
        return pRet;
}
// virtual
bool
CMSEndpoint::ProcessMessage(CMSMessage *pMsg)
{
        debug::printf("CMS", "Autoprocessing message S/N %d, CmdId %d\n", pMsg->GetSerialNo(),
                pMsg->GetCommandID());
        void *userdata;
        MESSAGEHANDLER handler = FindMessageHandler(pMsg->GetCommandID(), &userdata);
        if (!handler)
                handler = FindMessageHandler(0, &userdata);
        if (handler)
        {
                handler(pMsg, this, userdata);
                debug::printf("CMS", "Done with message S/N %d, CmdId %d\n",
                        pMsg->GetSerialNo(), pMsg->GetCommandID());
                return true;
        }
        debug::printf("CMS", "Couldn't find handler for message %d!\n", pMsg->GetCommandID());
        return false;
}
CMSMessage *
CMSEndpoint::TransactMessage(CMSMessage& sendmsg)
```

```
{
	bool error = false;
	CMSMessage *pReply = 0;
	::ResetEvent((HANDLE)cEvents.GetDispatchOKEvent());
	::SetEvent((HANDLE)cEvents.GetInTransactionEvent());

	if (WriteMessage(sendmsg))
	{
		while (!error)
		{
			if ((pReply = GetNextMessage()))
			{
				if (pReply->GetReplyToSerialNo() != sendmsg.GetSerialNo())
				{
					// put this message back into the queue
					//
					cMsgQueue.InsertMessage(pReply);
				}
				else
					break;
			}
			else
				error = true;
		}
	}
	::ResetEvent((HANDLE)cEvents.GetInTransactionEvent());
	::SetEvent((HANDLE)cEvents.GetDispatchOKEvent());

	return pReply;
}
void
CMSEndpoint::EnqueueMessage(CMSMessage *pMsg)
{
	cMsgQueue.InsertMessage(pMsg);
}
// virtual
MESSAGEHANDLER
CMSEndpoint::FindMessageHandler(unsigned short cmdid, void **userdata)
{
	return cMsgMap.FindMessageHandler(cmdid, userdata);
}
CMSMessage *
CMSEndpoint::TransformMessage(CMSMessage& msg)
{
	debug::printf("CMSEndpoint", "%p Transforming msg %p via %p...", this, &msg, AllocateSubclassMessage);
	CMSMessage *ret = AllocateSubclassMessage(&msg);
	debug::printf("CMSEndpoint", "got back %p\n", ret);
	if (ret)
		ret->Deserialize();
	return ret;
}
// virtual
CMSMessage *
CMSEndpoint::AllocateSubclassMessage(CMSMessage *pMsg)
{
	debug::printf("CMSEndpoint", "Oops!!! In CMSEndpoint(%p)::AllocateSubclassMessage!\n", this);
```

```
	return 0;
}
CMSMessage *
CMSEndpoint::ReadMessage(unsigned short NumUserObjects, unsigned long *UserObjects)
{
	// First, read a generic message from the endpoint
	//
	CMSMessage msg;
	if (msg.ReadFrom(cEndpoint, NumUserObjects, UserObjects))
	{
		// Now, transform it into the appropriate subclass
		//
		if (msg.GetCommandID() != C_NTFY_PING)
			debug::printf("MsgRead", "Transforming message S/N %d, CmdId %d\n",
				msg.GetSerialNo(), msg.GetCommandID());
		CMSMessage *pSubclassMessage = TransformMessage(msg);
		if (!pSubclassMessage)
		{
			pSubclassMessage = new CMSMessage(msg);
			debug::printf("CMSEndpoint", "Allocated CMSMessage %p because of no transformed
msg.\n",
				pSubclassMessage);
		}
		if (msg.GetCommandID() != C_NTFY_PING)
			debug::printf("MsgRead", "Done transforming.\n");
		return pSubclassMessage;
	}
	return 0;
}
void
CMSEndpoint::SetPingPeriod(unsigned long PeriodMilliseconds)
{
	Synchronize sync(cCriticalSection);

	cPingPeriod = PeriodMilliseconds;

	if (cSendPingThread)
	{
		ResetSendPingTimer();
		ResetReceivePingTimer(true);
	}
}
unsigned long
CMSEndpoint::GetPingPeriod(void)
{
	Synchronize sync(cCriticalSection);

	return cPingPeriod;
}
void
CMSEndpoint::ResetSendPingTimer(void)
{
	if (GetPingPeriod())
	{
		LARGE_INTEGER dueTime;
		dueTime.QuadPart = -((LONGLONG)GetPingPeriod() * (LONGLONG)10000);
```

```
                ::SetWaitableTimer((HANDLE)cEvents.GetPingSendTimerEvent(), &dueTime, 0L,
                        NULL, 0, FALSE);
        }
        else
        {
                // Disable timer
                //
                ::CancelWaitableTimer((HANDLE)cEvents.GetPingSendTimerEvent());
        }
}
void
CMSEndpoint::ResetReceivePingTimer(bool ResetDecay)
{
        if (GetPingPeriod())
        {
                LARGE_INTEGER dueTime;
                if (ResetDecay)
                        cPingDecayLevel = 0;
                dueTime.QuadPart = -((LONGLONG)((GetPingPeriod() << cPingDecayLevel) +
PING_SLACK_TIME) *
                        (LONGLONG)10000);
                ::SetWaitableTimer((HANDLE)cEvents.GetPingReceiveTimerEvent(), &dueTime, 0L,
                        NULL, 0, FALSE);
        }
        else
        {
                // Disable timer
                //
                ::CancelWaitableTimer((HANDLE)cEvents.GetPingReceiveTimerEvent());
        }
}
CMSMessage::SerialNumberT
CMSEndpoint::WriteMessage(CMSMessage& msg, unsigned short NumUserObjects,
        unsigned long *UserObjects)
{
        if (msg.GetCommandID() != C_NTFY_PING)
                debug::printf("MsgWrite", "Writing message S/N %d, CmdId %d\n",
                        msg.GetSerialNo(), msg.GetCommandID());
        ResetSendPingTimer();
   unsigned long t1 = ::GetTickCount();
        if (msg.WriteTo(cEndpoint, NumUserObjects, UserObjects))
        {
     debug::printf("PerfMeasure", "time it took to write message is %ld\n", ::GetTickCount() - t1);
                CMSMessage::SerialNumberT ret = msg.GetSerialNo();
                if (msg.GetCommandID() != C_NTFY_PING)
                        debug::printf("MsgWrite", "Wrote message S/N %d, CmdId %d\n",
                                msg.GetSerialNo(), msg.GetCommandID());
                return ret;
        }
        debug::printf("MsgWrite", "Failed writing message S/N %d, CmdId %d\n",
                msg.GetSerialNo(), msg.GetCommandID());
        return (CMSMessage::SerialNumberT)0;
}
void
CMSEndpoint::SetAutoDispatch(bool flag)
{
```

```
	Synchronize sync(cCriticalSection);

	if (flag)
		cFlags |= F_AUTODISPATCH;
	else
		cFlags &= ~F_AUTODISPATCH;
}
bool
CMSEndpoint::IsAutoDispatch(void)
{
	Synchronize sync(cCriticalSection);

	return (bool)((cFlags & F_AUTODISPATCH) != 0);
}
void
CMSEndpoint::SetIsClient(bool flag)
{
	Synchronize sync(cCriticalSection);

	if (flag)
		cFlags |= F_ISCLIENT;
	else
		cFlags &= ~F_ISCLIENT;
}
bool
CMSEndpoint::IsClient(void)
{
	Synchronize sync(cCriticalSection);

	return (bool)((cFlags & F_ISCLIENT) != 0);
}
void
CMSEndpoint::ShutdownThreads(void)
{
	if (cReadThread || cDispatchThread || cSendPingThread || cRecvPingThread)
	{
		debug::printf("CMS", "Setting event %lx\n", cEvents.GetTerminateThreadsEvent());
		::SetEvent((HANDLE)cEvents.GetTerminateThreadsEvent());
		while (cReadThread || cDispatchThread || cSendPingThread || cRecvPingThread);
	}
}
```

```
/*
 * CompositeMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:46 $
 */
static const char* id="$Id: CompositeMsg.cpp,v 1.2 2000/12/07 03:21:46 Mitsuru Oshima Exp $";
#include "CompositeMsg.h"
CompositeMessage::~CompositeMessage(void)
{
        std::list<VPPMessage *>::iterator iter = cMessages.begin();
        while (iter != cMessages.end())
        {
                delete *iter;
                iter++;
        }
}
// virtual
void
CompositeMessage::Serialize(void)
{
        VPPMessage::Serialize();
        InsertUshort((unsigned short)cMessages.size());

        std::list<VPPMessage *>::iterator iter = cMessages.begin();
        while (iter != cMessages.end())
        {
                InsertMessage(*iter);
                iter++;
        }
}
// virtual
void
CompositeMessage::Deserialize(void)
{
        VPPMessage::Deserialize();
        unsigned short size = ExtractUshort();
        for (unsigned short i = 0; i < size; i++)
        {
                VPPMessage *pMsg = new VPPMessage;
                ExtractMessage(pMsg, &cParse);
                cMessages.push_back(pMsg);
        }
}
```

```
/*
 * ConfigCmdLineList.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:22:15 $
 */
static const char* id="$Id: ConfigCmdLineList.cpp,v 1.2 2000/12/07 03:22:15 Mitsuru Oshima Exp $";
#include "ConfigCmdLineList.h"
void
ConfigCmdLineList::Initialize(int ac, char **av)
{
        for (int i = 1; i < ac; i++)
        {
                char *ptr = av[i];

                if (strchr(ptr, '='))
                {
                        char *s = new char[strlen(ptr) + 1];
                        strcpy(s, ptr);
                        list.push_back(s);
                }
        }
}
```

Document ID: 2000-0000-0000-0000

```
/*
 * ConfigFile.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:22:15 $
 */
static const char* id="$Id: ConfigFile.cpp,v 1.5 2000/12/07 03:22:15 Mitsuru Oshima Exp $";
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <iostream.h>
#include "ConfigMgr.h"
#include "ConfigFile.h"
#include "LogGroups.h"
ConfigFile::ConfigFile() :
        timerID(-1), currentListIndex(0), lastModTime(0), initialized(false)
{
}
ConfigFile::ConfigFile(const char *filename) :
        timerID(-1), currentListIndex(0), lastModTime(0), initialized(false)
{
        fileList[0].Initialize(filename);
        initialized = true;
}
ConfigFile::~ConfigFile()
{
// Trying to remove the timer causes a problem
//   if (timerID != -1)
//      GTimer::remove(timerID);
}
void
ConfigFile::UpdateConfig()
{
        gm::string configFile;
        if (!ConfigMgr::GetString(MTS_REGISTRY_BASE, "ConfigFile", configFile))
        {
                cerr << "No Config File specified" << endl;
                return;
        }
        struct _stat buffer;
        if (_stat(configFile, &buffer ) != 0)
        {
                cerr << "\n******* Cannot access Configeration File: \"" <<
      configFile << "\" *******\n" << endl;
    if (initialized)
                  UtilsLog::error("Cannot access Configeration File: \"%s\"", configFile);
        }
        else
        {
                if (!initialized || (lastModTime != buffer.st_mtime))
```

```
                {
                        lastModTime = buffer.st_mtime;

                        int newListIndex = currentListIndex ? 0 : 1;
                        fileList[newListIndex].CleanList();
                        fileList[newListIndex].Initialize(configFile.c_str());
                        currentListIndex = newListIndex;
                }
        }
  unsigned long interval = 10000;

        gm::string intervalString;
        if (ConfigMgr::GetString(MTS_REGISTRY_BASE ".Logging",
                "ConfigUpdateInterval", intervalString))
  {
    interval = ::atol(intervalString.c_str()) * 1000;
                if (interval < 5000)            // At least 5 seconds
                        interval = 5000;
  }
        timerID = GTimer::add(new TimerEventT<ConfigFile>(
                interval, this, ConfigFile::UpdateConfig));
}
void
ConfigFile::Initialize()
{
        if (!initialized)
        {
                UpdateConfig();
                initialized = true;
        }
        else
        {
                cerr << "Repeated Init of ConfigFile" << endl;
                UtilsLog::error("Repeated Init of ConfigFile");
        }
}
unsigned short
ConfigFile::GetNumLines(void)
{
        return fileList[currentListIndex].GetNumLines();
}
const char *
ConfigFile::GetLine(unsigned short ndx)
{
        return fileList[currentListIndex].GetLine(ndx);
}
bool
ConfigFile::GetString(const char *key, gm::string& retString)
{
        return fileList[currentListIndex].GetString(key, retString);
}
```

```
/*
 * ConfigFileList.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:22:15 $
 */
static const char* id="$Id: ConfigFileList.cpp,v 1.2 2000/12/07 03:22:15 Mitsuru Oshima Exp $";
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include "ConfigMgr.h"
#include "ConfigFileList.h"
void
ConfigFileList::Initialize()
{
        gm::string filename;
        ConfigMgr::GetString(MTS_REGISTRY_BASE, "ConfigFile", filename);
        Initialize(filename.c_str());
}
void
ConfigFileList::Initialize(const char *filename)
{
        FILE *fp = fopen(filename, "rt");
        if (fp)
        {
                char str[256];
                while (fgets(str, 256, fp))
                {
                        if (str[strlen(str) - 1] == 0xa)
                                str[strlen(str) - 1] = 0;
                        if (str[strlen(str) - 1] == 0xd)
                                str[strlen(str) - 1] = 0;
                        if ((strlen(str) == 0) || (str[0] == '@'))
                                continue;
                        char *s = new char[strlen(str) + 1];
                        strcpy(s, str);
                        list.push_back(s);
                }
                fclose(fp);
        }
}
```

```
/*
 * ConfigInternalList.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.6 $
 * $Date: 2000/12/07 03:22:16 $
 */
static const char* id="$Id: ConfigInternalList.cpp,v 1.6 2000/12/07 03:22:16 Mitsuru Oshima Exp $";
#include "ConfigInternalList.h"
char *ConfigInternalList::configString[] = {
        "MTS.ConfigFile=C:\\Config.mts",
        "MTS.ClientClusterName=MagicTalk",
        "MTS.ServerClusterName=MagicTalk",
        "MTS.DefaultRecordingsDir=.\\",
        "MTS.Media Service Provider=Dialogic Voice Device",
        "MTS.PromptRoot=c:\\prompts",
        "MTS.Speech Service Provider=Nuance",
        "MTS.Telephony Service Provider=GlobalCall",
        "MTS.TTS Service Provider=Acuvoice",
        "MTS.Groups.NumGroups=2",
        "MTS.Groups.Group1.Name=TestGroup",
        "MTS.Groups.Group2.Name=AgentNotification",
        "MTS.Logging.DefaultLogLevel=INFO",
        "MTS.Logging.EnableWholeCallRecording=0",
        "MTS.Logging.MTSLogDir=c:\\",
        "MTS.Logging.MTSLogFile=mts.out",
        "MTS.Logging.DebugGroups=",
        "MTS.Logging.ReparseIntervalSeconds=30",
        "MTS.Logging.ConfigUpdateInterval=10",
        "MTS.Media Service Providers.Dialogic Voice Device.Library=D:\\genmagic\\bin\\dialogicMSP.dll",
        "MTS.Speech Service Providers.Nuance.Library=D:\\genmagic\\bin\\NuanceSSP70.dll",
        "MTS.Speech Service Providers.Nuance.LogRecordingFilename=false",
        "MTS.Speech Service
Providers.Nuance.PackagesSpecificationFile=c:\\nuance\\v7.0.1\\data\\nuance.packages",
        "MTS.Speech Service Providers.Nuance.ParametersFile=c:\\nuance\\v7.0.1\\data\\nuance.parameters",
        "MTS.Speech Service Providers.Nuance.Database.dbclass=dgdb",
        "MTS.Speech Service Providers.Nuance.Database.dbname=mts",
        "MTS.Telephony Service Providers.GlobalCall.IncomingLines=0-22",
        "MTS.Telephony Service Providers.GlobalCall.OutgoingLines=0-22",
        "MTS.Telephony Service Providers.GlobalCall.Library=D:\\genmagic\\bin\\GlobalCallTSP.dll",
        "MTS.TTS Service Providers.Acuvoice.Library=D:\\genmagic\\bin\\AcuvoiceTTSSP.dll",
        ""      // Termination
};
void
ConfigInternalList::Initialize()
{
        for (int i = 0; *configString[i]; i++)
        {
                int len = strlen(configString[i]);
                char *s = new char[len + 1];
                strcpy(s, configString[i]);
```

```
            list.push_back(s);
        }
    }
```

Downloaded by [redacted]

```
/*
 * ConfigList.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:22:16 $
 */
static const char* id="$Id: ConfigList.cpp,v 1.3 2000/12/07 03:22:16 Mitsuru Oshima Exp $";
//#include <string.h>
//#include <sys/stat.h>
#include "ConfigList.h"
#include "LogGroups.h"
ConfigList::ConfigList()
{
}
ConfigList::~ConfigList()
{
        CleanList();
}
void
ConfigList::CleanList()
{
        std::list<char *>::iterator iter = list.begin();
        while (iter != list.end())
        {
                char *s = *iter;
                delete [] s;
                iter++;
        }
        list.clear();
}
unsigned short
ConfigList::GetNumLines(void)
{
        return list.size();
}
const char *
ConfigList::GetLine(unsigned short ndx)
{
        std::list<char *>::iterator iter = list.begin();
        unsigned short i = 0;
        while (i < ndx && iter != list.end())
        {
                i++;
                iter++;
        }
        if (iter != list.end())
                return *iter;
        return 0;
}
bool
```

```
ConfigList::GetString(const char *key, gm::string& retString)
{
	retString = "";
	gm::string tmpKey = key;
	tmpKey += "=";
	int len = strlen(tmpKey);
	std::list<char *>::iterator iter = list.begin();
	while (iter != list.end())
	{
		char *strPtr = (*iter);
		if (strnicmp(tmpKey, strPtr, len) == 0)		// Ignoring case
		{
			strPtr = strPtr + len;
			retString = strPtr;
			return true;
		}
		iter++;
	}
	return false;
}
```

```
/*
 * ConfigMgr.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.6 $
 * $Date: 2000/12/07 03:22:16 $
 */
static const char* id="$Id: ConfigMgr.cpp,v 1.6 2000/12/07 03:22:16 Mitsuru Oshima Exp $";
#include <windows.h>
#include "ConfigMgr.h"
#include "LogGroups.h"
#ifdef USE_REGISTRY   /**** Registry ****/
#include "RegistryMgr.h"
#endif
bool ConfigMgr::initialized = false;
ConfigInternalList ConfigMgr::internalList;
ConfigFile ConfigMgr::fileList;
ConfigCmdLineList ConfigMgr::cmdLineList;
// Temporary
void replaceChar(char *string, char orig, char replacement)
{
        for (char *charPtr = string; *charPtr; charPtr++)
        {
                if (*charPtr == orig)
                        *charPtr = replacement;
        }
}
bool
ConfigMgr::GetString(const char *parent, const char *key, gm::string& retString)
{
#ifdef USE_REGISTRY   /**** Registry ****/
        char parentStr[512], keyStr[512];
        strcpy(parentStr, parent);
        replaceChar(parentStr, '.', '\\');
        strcpy(keyStr, key);
        replaceChar(keyStr, '.', '\\');
        return RegistryMgr::GetString(parentStr, keyStr, retString);
#else   /**** Config File ****/
        if (!initialized)
        {
//              UtilsLog::error("ConfigMgr not initialized");
                ConfigMgr::Initialize(0, NULL);
        }
        gm::string fullKey = parent;
        int len = strlen(parent);
        if (len)
        {
                if (*(parent + len - 1) == '\\')
                        fullKey.resize(len - 1);
                if (*(parent + len - 1) != '.')
                        fullKey += '.';
```

```
        }
        fullKey += key;
        // Temporary: Replace '\' with '.'
        char *keyStr = new char[fullKey.size() + 1];
        strcpy(keyStr, fullKey.c_str());
        for (char *charPtr = keyStr; *charPtr; charPtr++)
        {
                if (*charPtr == '\\')
                        *charPtr = '.';
        }
        bool result =  (cmdLineList.GetString(keyStr, retString) ||
                        fileList.GetString(keyStr, retString) ||
                        internalList.GetString(keyStr, retString));
        if (!result)
        {
//printf("**** Couldn't find: \"%s\" Config Key ****\n", keyStr);
// Logging may not be ready
//              UtilsLog::info("**** Couldn't find: \"%s\" Config Key ****\n", keyStr);
        }
        delete [] keyStr;
        return result;
#endif  /**** Config File ****/
}
bool
ConfigMgr::GetInteger(const char *parent, const char *key, int &value)
{
        value = 0;
        gm::string valueStr;
        if (!GetString(parent, key, valueStr))
                return false;
// Should check for good interger

        value = ::atoi(valueStr.c_str());
        return true;
}
bool
ConfigMgr::GetBoolean(const char *parent, const char *key, bool &value)
{
        value = false;
        gm::string valueStr;
        if (!GetString(parent, key, valueStr))
                return false;
        if (valueStr.compare("true") == 0)
                value = true;
        else if (valueStr.compare("false") != 0)
                UtilsLog::error("Bad config value of %s for key of %s", valueStr.c_str(), key);
        return true;
}
void
ConfigMgr::Initialize(int ac, char **av)
{
#ifndef USE_REGISTRY /**** Not Registry ****/
        initialized = true;
        internalList.Initialize();
        cmdLineList.Initialize(ac, av);
        fileList.Initialize();
```

```
#endif
}
```

Document1 - Microsoft Word

```
/*
 * ConnectionListener.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:19:46 $
 */
static const char* id="$Id: ConnectionListener.cpp,v 1.2 2000/12/07 03:19:46 Mitsuru Oshima Exp $";
#include "ConnectionListener.h"
#include "ConnectionMgr.h"
ConnectionListener::~ConnectionListener()
{
   ConnectionMgr::removeConnectionListener(this);
}
```

Docket No. 3000.2.2

```
/*
 * ConnectionMgr.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:19:46 $
 */
static const char* id="$Id: ConnectionMgr.cpp,v 1.4 2000/12/07 03:19:46 Mitsuru Oshima Exp $";
#include "ConnectionMgr.h"
#include "ConnectionWatcher.h"
#include "SessionMgr.h"
#include "MTSServerImpl.h"
#include "invui_c.h"
#include <VPPUtils/Log.h>
#include "MTSServer.h"
class PeerTable : public ConnectionMgr::ConnectionTable {
public:
  PeerTable() : Hashtable<char*, ConnectionMgr::ConnectionListenerList*>() {}
  virtual ~PeerTable() {
    clear();
  }
  int _hash(char* seq) {
    return seq[0] + seq[ strlen(seq)-1];
  }
  char* _copy(char* value) {
    char* copy = new char[strlen(value)+1];
    strcpy(copy, value);
    return copy;
  }
  bool _keyEquals(char* k1, char* k2) {
    return strcmp(k1, k2) == 0;
  }
  bool _valueEquals(ConnectionMgr::ConnectionListenerList* v1,
            ConnectionMgr::ConnectionListenerList* v2) {
    return v1 == v2;
  }
  void _deleteKey(char* key) { // no output
    delete[] key;
  }
};
char*
ConnectionMgr::ConnectionTable::NULL_KEY = NULL;
ConnectionMgr::ConnectionListenerList*
ConnectionMgr::ConnectionTable::NULL_VALUE = NULL;
ConnectionMgr::ConnectionTable*
ConnectionMgr::table = new PeerTable();
void
ConnectionMgr::addConnectionListener(invui::Pingable_ptr ping,
                     ConnectionListener* listener)
{
  Synchronized sync(table);
```

```
    MTSServerImpl* mtsserver = SessionMgr::GetSessionMgr()->GetMTSServerImpl();
    CORBA::ORB_ptr orb = mtsserver->GetORB();
    char* id = orb->object_to_string(ping);
    ConnectionMgr::ConnectionListenerList* list = table->get(id);
    if (list == NULL) {
        list = new ConnectionListenerList();
        MTSLibLog::info("adding listener list %p for %s", list, id);
        table->put(id, list);
        (new ConnectionWatcher(ping, id))->start();
    }
    CORBA::string_free(id);
    if (list->contains(listener)) {
        return;
    }
    list->push(listener);
}
void
ConnectionMgr::removeConnectionListener(ConnectionListener* listener)
{
    Synchronized sync(table);
    MTSLibLog::info("removing listener %p", listener);
    ConnectionMgr::ConnectionTable::Iterator itr = table->iterator();
    while (itr.next()) {
        ConnectionMgr::ConnectionListenerList* list = itr.value();
        while (list->contains(listener)) {
            list->remove(listener);
        }
    }
}
void
ConnectionMgr::shutdown(char* id) {
    ConnectionMgr::ConnectionListenerList* list = ConnectionMgr::table->remove(id);
    MTSLibLog::info("removing listener list %p", list);
            if (list == NULL) {
                    return;
            }
    ConnectionListener* listener;
    while ( (listener = list->pop()) != NULL)
    {
        MTSLibLog::info("disconnecting listener %p", listener);
        listener->Disconnected();
    }
    delete list;
}
```

```
/*
 * InsertDynamicGrammarResultNtfy.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:50 $
 */
static const char* id="$Id: InsertDynamicGrammarResultNtfy.cpp,v 1.2 2000/12/07 03:21:50 Mitsuru Oshima Exp $";
#include "InsertDynamicGrammarResultNtfy.h"
// virtual
void
InsertDynamicGrammarResultNotification::Serialize(void)
{
        DynamicGrammarResultNotification::Serialize();
        InsertString(cGrammarName);
}
// virtual
void
InsertDynamicGrammarResultNotification::Deserialize(void)
{
        DynamicGrammarResultNotification::Deserialize();
        ExtractString(cGrammarName);
}
```

```
/*
 * CreateDynamicGrammarMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:47 $
 */
static const char* id="$Id: CreateDynamicGrammarMsg.cpp,v 1.2 2000/12/07 03:21:47 Mitsuru Oshima Exp $";
#include "CreateDynamicGrammarMsg.h"
// virtual
void
CreateDynamicGrammarMessage::Serialize(void)
{
        DynamicGrammarMessage::Serialize();
        InsertString(cGSL);
}
// virtual
void
CreateDynamicGrammarMessage::Deserialize(void)
{
        DynamicGrammarMessage::Deserialize();
        ExtractString(cGSL);
}
```

Document ID: 2000P05201

```
/*
 * CriticalSection.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:22:17 $
 */
static const char* id="$Id: CriticalSection.cpp,v 1.2 2000/12/07 03:22:17 Mitsuru Oshima Exp $";
#include "CriticalSection.h"
CriticalSection::CriticalSection(void)
{
        ::InitializeCriticalSection(&cCriticalSection);
}
CriticalSection::~CriticalSection(void)
{
        ::DeleteCriticalSection(&cCriticalSection);
}
void
CriticalSection::Enter(void)
{
        ::EnterCriticalSection(&cCriticalSection);
}
void
CriticalSection::Leave(void)
{
        ::LeaveCriticalSection(&cCriticalSection);
}
```

```
/*
 * DefaultLog.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.24 $
 * $Date: 2000/12/07 03:22:17 $
 */
static const char* id="$Id: DefaultLog.cpp,v 1.24 2000/12/07 03:22:17 Mitsuru Oshima Exp $";
//
// Registry stuff for logs:
// "Software\General Magic\MTS\Logging\MTSLogDir"
//                  Directory to put the mts_DATE.out file in..
// "Software\General Magic\MTS\Logging\DefaultLogLevel" =
//                  "ERROR", "WARNING", "INFO", "DIAG", or "DEBUG"
// "Software\General Magic\MTS\Logging\DebugGroups" = names of
//                  debug groups that should be enabled for debug
//                  log output.
// "Software\General Magic\MTS\Logging\ReparseIntervalSeconds" =
//                  interval in seconds between rechecking
//                  registry settings for DefaultLogLevel and
//                  DebugGroups.
//
#include <sys/timeb.h>
#include <stdio.h>
#include <io.h>
#include <iostream.h>
#include <time.h>
#include <stdarg.h>
#include "DefaultLog.h"
#include "Timer.h"
#include "GMString"
#include "ConfigMgr.h"
#include "Synchronized.h"
static int every = 1;
static int flush_count = every;
class LogFile {
public:
        LogFile(const char *directory, const char *filename);
        ~LogFile();
        bool SetOutputFilename(const char* fileName, bool overwrite);
        void PutString(const char *buf);
private:
        void NewLogFile();
        void CloseOldFile();
        gm::string MakeTimestampFilename(const char* directory, const char* fileName);
        FILE* filePtr;
        FILE* oldFilePtr;
        bool    canClose;
  int           fileDay;    // Day last log file was created.
        int             outCount;
        gm::string logFileDir;
```

```
        gm::string logFileName;
        Timer::TimerID openTimerID;
        Timer::TimerID closeTimerID;
};
LogFile::LogFile(const char *directory, const char *filename) :
                oldFilePtr(NULL), filePtr(stderr),
                canClose(false), fileDay(400), outCount(0),
                openTimerID(-1), closeTimerID(-1) {
        logFileDir = directory;
        logFileName = filename;
        NewLogFile();
}
LogFile::~LogFile() {
        GTimer::remove(openTimerID);
        if (closeTimerID != -1) {
                GTimer::remove(closeTimerID);
        }
}
void
LogFile::NewLogFile()
{
   struct _timeb tb;
   _ftime(&tb);
   struct tm *time = localtime(&tb.time);

        if (fileDay != time->tm_mday) {
                gm::string fileName = logFileName.c_str();
                if (SetOutputFilename(fileName.c_str(), false)) {
                        fileDay = time->tm_mday;
                }
   }
        unsigned long secTillMidnight = (24 * 60 * 60) + 1 -
                ((((time->tm_hour) * 60) + time->tm_min) * 60) + time->tm_sec;
        openTimerID = GTimer::add(new TimerEventT<LogFile>(
                secTillMidnight * 1000, this, LogFile::NewLogFile));
}
bool
LogFile::SetOutputFilename(const char* fileName, bool overwrite) {
        bool status = true;
        gm::string pathName =
                LogFile::MakeTimestampFilename(logFileDir, fileName).c_str();
        if (overwrite || (strcmp(logFileName.c_str(), pathName.c_str()) != 0))
        {
                FILE* tempFilePtr = fopen(pathName, overwrite ? "wt" : "a+t");
                if (tempFilePtr != NULL) {
                        if (filePtr != stderr) {
                                oldFilePtr = filePtr;
                                canClose = false; // Other threads may be writting to file right now.
                                CloseOldFile();
                        }
                        filePtr = tempFilePtr;
      logFileName = fileName;
                }
    else
    {
                cerr << "Cannot open new log file\a\a\a\a\a\a!" << endl;
```

```
        status = false;
      }
        }
        return status;
}
void
LogFile::CloseOldFile() {
        if (oldFilePtr != NULL) {
                if (canClose) {
                        closeTimerID = -1;
                        fclose(oldFilePtr);
                        oldFilePtr = NULL;
                }
                else
                {
                        closeTimerID = GTimer::add(new TimerEventT<LogFile>(
                                30 * 1000, this, LogFile::CloseOldFile));
                }
        }
}
gm::string
LogFile::MakeTimestampFilename(const char* directory, const char* fileName) {
   gm::string tempFileName = "";
   gm::string timestampFileName = "";

        // If filename doesn't already contain full path, put in directory.
        if ((strchr(fileName, ':') == NULL) &&
                (fileName[0] != '\\') &&
                (fileName[0] != '/')) {
                //  Does directory exist (and is writable)
                if (::access(directory, 2) != 0) {
                        tempFileName = directory + '\\';
                }
                else {
                        cerr << "Bad Log file directory:\a\a\a\a\a\a" << directory << endl;
                }
        }
        tempFileName += fileName;

        //
        //  extract the extension of the file (to be added later)
   //
   int bufLen = tempFileName.size() + 20;
        gm::string tempNameExt = "";
        int pos;
        if ((pos = tempFileName.rfind('.')) != gm::string::npos)
        {
                tempNameExt = (char *)(tempFileName.c_str() + pos);
                tempFileName.erase(pos);
        }
   struct _timeb tb;
   _ftime(&tb);
   struct tm *time = localtime(&tb.time);
   char* buf = new char[bufLen];
   ::_snprintf(buf, bufLen - 1, "%s_%d_%d_%0.2d%s",
      tempFileName.c_str(),
```

```
        time->tm_mon + 1,
        time->tm_mday,
        time->tm_year%100,
        tempNameExt.c_str()
        );
    buf[bufLen - 1] = '\0';
    timestampFileName = buf;
    delete buf;
            return timestampFileName;
}
void
LogFile::PutString(const char *buf)
{
            outCount++;
            ::fputs(buf, filePtr);
            ::fflush(filePtr);
            outCount--;
            // Old output file can be closed anytime after finding
            // that nothing is in the process of being written.
            // Any subsequent writes go to the new file.
            if (outCount == 0) {
                        canClose = true;
            }
}
/***
 * Default Log
 */
LogFile *DefaultLog::logFile = NULL;
void
DefaultLog::Initialize(const char *filename, bool overwrite)
{
            if (logFile != NULL)
                        return;
            gm::string logFileDir, logFileName;

            // Get Logging Directory
    if (!ConfigMgr::GetString(MTS_REGISTRY_BASE ".Logging", "MTSLogDir", logFileDir)) {
        logFileDir = "C:\\";
    }
    if (filename == NULL || *filename == NULL)
                        logFileName = "log.out";
    else
        logFileName = filename;
    logFile = new LogFile(logFileDir.c_str(), logFileName.c_str());
    atexit(DefaultLog::Shutdown);  // Do we need to do this??
}
void
DefaultLog::Shutdown()
{
    if (logFile != NULL) {
                logFile->~LogFile();
                logFile = NULL;
    }
}
DefaultLog::DefaultLog(const char* n)
: name(n) {
```

```
}
bool
DefaultLog::SetOutputFilename(const char* fileName, bool overwrite) {
        return logFile->SetOutputFilename(fileName, overwrite);
}
void
DefaultLog::print(const char* type, const char *fmt, const va_list vl)
{
   struct _timeb tb;
   _ftime(&tb);
   struct tm *time = localtime(&tb.time);
   char buf[256];
   ::_snprintf(buf, sizeof(buf), "%s|%.2d.%.2d|%.2d:%.2d:%.2d.%.3d|%s|",
           type,
           time->tm_mon + 1,
           time->tm_mday, time->tm_hour, time->tm_min, time->tm_sec,
           tb.millitm,
           name);
   buf[sizeof(buf) - 2] = '\0';          // Room for \n
   int len = strlen(buf);
   if (::_vsnprintf(&buf[len], (sizeof(buf) - 2) - len, fmt, vl) < 0) {
      char *bufA = new char[65535];
      strncpy(bufA, buf, len);
      ::_vsnprintf(&bufA[len], 65535 - 2 - len, fmt, vl);
      bufA[65535 - 2] = '\0';
      len = strlen(bufA);
      bufA[len] = '\n';
      bufA[len+1] = '\0';
      logFile->PutString(bufA);
      delete bufA;
   }
   else
   {
      len = strlen(buf);
      buf[len] = '\n';
      buf[len+1] = '\0';
      logFile->PutString(buf);
   }
}
inline bool
DefaultLog::errorEnabled() {
   return (enableMask & LOGMASK_ERROR) != 0;
}
inline bool
DefaultLog::warningEnabled() {
   return (enableMask & LOGMASK_WARNING) != 0;
}
inline bool
DefaultLog::infoEnabled() {
   return (enableMask & LOGMASK_INFO) != 0;
}
inline bool
DefaultLog::diagEnabled() {
   return (enableMask & LOGMASK_DIAG) != 0;
}
inline bool
```

```
DefaultLog::debugEnabled() {
    return (enableMask & LOGMASK_DEBUG) != 0;
}
void
DefaultLog::va_error(const char *fmt, const va_list vl) {
    if (errorEnabled()) {
        print("ERR ", fmt, vl);
    }
}
void
DefaultLog::va_warning(const char *fmt, const va_list vl) {
    if (warningEnabled()) {
        print("WARN", fmt, vl);
    }
}
void
DefaultLog::va_info(const char *fmt, const va_list vl) {
    if (infoEnabled()) {
        print("INFO", fmt, vl);
    }
}
void
DefaultLog::va_diag(const char *fmt, const va_list vl) {
    if (diagEnabled()) {
        print("DIAG", fmt, vl);
    }
}
void
DefaultLog::va_debug(const char *fmt, const va_list vl) {
    if (debugEnabled()) {
        print("DEBG", fmt, vl);
    }
}
void
DefaultLog::error(const char *fmt, ...) {
    if (errorEnabled()) {
        va_list vl;
        va_start(vl, fmt);
        print("ERR ", fmt, vl);
    }
}
void
DefaultLog::warning(const char *fmt, ...) {
    if (warningEnabled()) {
        va_list vl;
        va_start(vl, fmt);
        print("WARN", fmt, vl);
    }
}
void
DefaultLog::info(const char *fmt, ...) {
    if (infoEnabled()) {
        va_list vl;
        va_start(vl, fmt);
        print("INFO", fmt, vl);
    }
```

```
}
void
DefaultLog::diag(const char *fmt, ...) {
   if (diagEnabled()) {
      va_list vl;
      va_start(vl, fmt);
      print("DIAG", fmt, vl);
   }
}
void
DefaultLog::debug(const char *fmt, ...) {
   if (debugEnabled()) {
      va_list vl;
      va_start(vl, fmt);
      print("DEBG", fmt, vl);
   }
}
void
DefaultLog::setMask(unsigned int m) {
   enableMask = m;
}
/**
 * Default Log Factory
 */
#define  NULL_MASK -1
class MaskTable : public Hashtable<char*, unsigned int> {
public:
   MaskTable() : Hashtable<char*, unsigned int>() {}
   virtual ~MaskTable() {
      clear();
   }
   int _hash(char* value) {
      int h = strlen(value) + value[0];
      return h;
   }
   char* _copy(char* value) {
      char* copy = new char[strlen(value)+1];
      strcpy(copy, value);
      return copy;
   }
   bool _keyEquals(char* k1, char* k2) {
      return strcmp(k1, k2) == 0;
   }
   bool _valueEquals(unsigned int v1, unsigned int v2) {
      return v1 == v2;
   }
   void _deleteKey(char* key) { // no output
      delete[] key;
   }
};
char*
Hashtable<char*, unsigned int>::NULL_KEY = 0;
unsigned int
Hashtable<char*, unsigned int>::NULL_VALUE = NULL_MASK;
DefaultLogFactory::DefaultLogFactory() :
        cAllPresent(false), defaultEnableMask(0),
```

```
        currentConfig(""), mask_table(NULL), timerID(-1)
{
  mask_table = new MaskTable();
  updateLogLevels();
}
DefaultLogFactory::~DefaultLogFactory()
{
  if (timerID != -1)
  {
    GTimer::remove(timerID);
  }
  if (mask_table != NULL)
  {
    delete mask_table;              // Add back when delete of hash table works
  }
}
VPPLog*
DefaultLogFactory::createLog(const char* name) {
  DefaultLog* log = new DefaultLog(name);
  if (cAllPresent == TRUE) {
    log->setMask(DefaultLog::LOGMASK_ALL);
  } else {
    log->setMask(GetMaskFor((char*)name));
  }
  logs.push_back(log);
  return log;
}
unsigned int
DefaultLogFactory::GetMaskFor(const char* name) {
  if (cAllPresent) {
    return DefaultLog::LOGMASK_ALL;
  }
  else {
    unsigned int mask = mask_table->get((char*)name);
    if (mask != NULL_MASK)
      return mask;
    else
      return defaultEnableMask;
  }
}
unsigned int
DefaultLogFactory::getRegistryMask(const char* name, unsigned int defaultMask) {
        unsigned int mask = defaultMask;
  gm::string regstr;
  if (ConfigMgr::GetString(MTS_REGISTRY_BASE ".Logging",
                  name, regstr)) {
    if (strcmp("ERROR", regstr.c_str()) == 0) {
      mask = DefaultLog::LOGMASK_ERROR;
    } else if (strcmp("WARNING", regstr.c_str()) == 0) {
      mask = DefaultLog::LOGMASK_ERROR |
        DefaultLog::LOGMASK_WARNING;
    } else if (strcmp("INFO", regstr.c_str()) == 0) {
      mask = DefaultLog::LOGMASK_ERROR |
        DefaultLog::LOGMASK_WARNING |
        DefaultLog::LOGMASK_INFO;
    } else if (strcmp("DIAG", regstr.c_str()) == 0) {
```

```
        mask = DefaultLog::LOGMASK_ERROR |
          DefaultLog::LOGMASK_WARNING |
          DefaultLog::LOGMASK_INFO |
          DefaultLog::LOGMASK_DIAG;
      } else if (strcmp("DEBUG", regstr.c_str()) == 0) {
        mask = DefaultLog::LOGMASK_ALL;
      }
    }
    return mask;
}
void
DefaultLogFactory::updateLogLevels() {
        gm::string regstr;
        {
                Synchronized sync(this);
                // UtilsLog::info("updating log configuration");


                unsigned int mask = getRegistryMask("DefaultLogLevel", DefaultLog::LOGMASK_ERROR |
                        DefaultLog::LOGMASK_WARNING | DefaultLog::LOGMASK_INFO);
                if ((ConfigMgr::GetString(MTS_REGISTRY_BASE ".Logging", "DebugGroups", regstr) &&
                        strcmp(currentConfig.c_str(), regstr.c_str())) ||
                        (defaultEnableMask != mask)) {
                        cerr << "updating log configuration" << endl;

                        defaultEnableMask = mask;
                        currentConfig = regstr;

                        mask_table->clear();
                        char *str = strdup(regstr.c_str());
                        char *substr = 0;
                        while ((substr = strtok(substr ? 0 : str, ", \t;"))) {
                                if (strcmp(substr, "All")) {
                                        unsigned int mask = getRegistryMask(substr,
DefaultLog::LOGMASK_ALL);
                                        mask_table->put(substr, mask);
                                }
                                else {
                                        cAllPresent = TRUE;
                                }
                        }
                        free(str);

                        // set enableMask for all log objects created so far
                        std::list<DefaultLog *>::iterator iterlog = logs.begin();
                        while (iterlog != logs.end()) {
                                DefaultLog* l = *iterlog;
                                l->setMask(GetMaskFor((char*)l->name));
                                iterlog ++;
                        }
                }
        }
    int interval;

    if (ConfigMgr::GetInteger(MTS_REGISTRY_BASE ".Logging",
                "ReparseIntervalSeconds", interval))
```

```
    {
        interval = interval * 1000;
                if (interval < 5000)          // At least 5 seconds
                        interval = 5000;
    }
        else
        {
                interval = 30000;
        }
        timerID = GTimer::add(new TimerEventT<DefaultLogFactory>(
                interval, this, DefaultLogFactory::updateLogLevels));
}
```

Document ID: 2.B.2.2.2.2.2.0

```
/*
 * DialogicMChannel.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.81 $
 * $Date: 2000/12/07 03:25:07 $
 */
static const char* id="$Id: DialogicMChannel.cpp,v 1.81 2000/12/07 03:25:07 Mitsuru Oshima Exp $";
#define _WIN32_WINNT 0x0400
#include <windows.h>
#include <process.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include "DialogicMChannel.h"
#include "VPPUtils/Synchronize.h"
#include "VPPMsg/PlayingNtfy.h"
#include "VppMsg/RecordingDoneNtfy.h"
#include "MTSLib/PromptSegment.h"
#include "MTSLib/SessionMgr.h"
#include "MTSLib/MTSServer.h"
#include "MTSLib/SessionNode.h"
#include "MTSLib/TTSChannel.h"
//#include "../../GlobalCall/TSP/GlobalCallTSP.h"
bool DialogicMChannel::cFirstCallback=true;
unsigned long DialogicMChannel::cCallbackTime = 0L;
class DLGCMSPLog : public LogT<DLGCMSPLog> {
  DLGCMSPLog() : LogT<DLGCMSPLog>("DLGC_MSP") {};
  static DLGCMSPLog log; // for initialization
};
DLGCMSPLog DLGCMSPLog::log;
DialogicMChannel::DialogicMChannel(ServiceProvider *pServiceProvider, const char *logType, int id)
        : MediaChannel(pServiceProvider, logType, id, true),
          cCurFilePos(0L), cpPlayFile(NULL),
          cDMChanState(S_Idle), cSilenceBytes(0L),
          cRecordStartedTime(0L), cRecordingStoppedTime(0L),
    cPlayingStoppedTime(0L), cBosTimeout(-1), cTmsTimeout(-1), cEosTimeout(-1)
{
        cHdl = -1;
        cSilencePromptTimer = ::CreateWaitableTimer(NULL, false, NULL);
        cStopTimerEvent = ::CreateEvent(NULL, true, false, NULL);  // must be manually reset!
        cStreamPlayReadMutex = ::CreateMutex(NULL, false, NULL);
}
DialogicMChannel::~DialogicMChannel(void)
{
        ::CloseHandle(cStreamPlayReadMutex);
        ::CloseHandle(cStopTimerEvent);
        ::CloseHandle(cSilencePromptTimer);
}
bool
DialogicMChannel::Setup(const gm::string& chname)
```

```
{
        cHdl = dx_open(chname, 0);
        if (cHdl != -1)
        {
//              if (!((GlobalCallTSP *)SessionMgr::GetSessionMgr()->GetMTSServer()-
>GetTelephonyServiceProvider())->MonitorDevice(cHdl))
//              {
//                      error("Cannot monitor DialogicMChannel %s", chname.c_str());
//                      return false;
//              }
                // Unfortunate hack- we have to set all the flags to account for the fact
                // that there is no dx_getevtmsk
                //
                dx_setevtmsk(cHdl, DM_LCOFF | DM_LCON | DM_RINGS | DM_RNGOFF |
                        DM_SILOF | DM_SILON | DM_WINK | DM_DIGITS | DM_LCREV);
//              sr_enbhdlr(cHdl, TDX_PLAY, dx_handler);
//              sr_enbhdlr(cHdl, TDX_RECORD, dx_handler);
//              sr_enbhdlr(cHdl, TDX_ERROR, dx_handler);
//              sr_enbhdlr(cHdl, TDX_CST, dx_handler);
                long ts;
                SC_TSINFO tsinfo;
                tsinfo.sc_numts = 1;
                tsinfo.sc_tsarrayp = &ts;
                if (dx_getxmitslot(cHdl, &tsinfo) == -1)
                {
                        error("dx_getxmitslot failed!");
                        return false;
                }
                diag("Media channel %s is transmitting on timeslot %d",
            ATDV_NAMEP(cHdl), (unsigned short)ts);
                cTxTimeslot = (unsigned short)ts;
        }
        return (bool)(cHdl != -1);
}
long
DialogicMChannel::dx_handler(unsigned long evhandle)
{
  DLGCMSPLog::debug("calling sr_getevtdev");
  int chdesc = sr_getevtdev(evhandle);
  DLGCMSPLog::debug("called sr_getevtdev, calling FindChannelForDescriptor");
        DialogicMChannel *pChannel = DialogicMSP::GetInstance()->
                FindChannelForDescriptor(chdesc);
  DLGCMSPLog::debug("calling HandleEvent");
        if (!pChannel || !pChannel->HandleEvent(evhandle))
    DLGCMSPLog::warning("Unhandled Dialogic event 0x%X on device %s (%d)",
                                sr_getevttype(evhandle), ATDV_NAMEP(chdesc), chdesc);
        return 1;
}
// virtual
bool
DialogicMChannel::HandleEvent(unsigned long evhandle)
{
        int chdesc;
        long EventCode;
        long Reason;
  debug("Calling sr_getevtdev");
```

```
        chdesc = sr_getevtdev(evhandle);
    debug("Called sr_getevtdev");
        EventCode = sr_getevttype(evhandle);
    debug("Called sr_getevttype");
        switch(EventCode)
        {
                case TDX_CST:
                {
                        DX_CST *cst = (DX_CST *)sr_getevtdatap(evhandle);
                        switch(cst->cst_event)
                        {
                                case DE_DIGITS:
                            // diag("Media Channel %s (%d) got digit %d", ATDV_NAMEP(cHdl), cHdl,
                                        diag("got digit %d",
                             cst->cst_data);
                                        NotifyDTMF(cst->cst_data);
                                        break;
                        }
                }
                break;
                case TDX_PLAY:
                {
                        Reason = ATDX_TERMMSK(chdesc);
                        debug("TDX_PLAY - play is done, reason = %ld. Time it took to stop playing : %ld.",
Reason, ::GetTickCount() - cDoStopTime);
                        bool bNotifyPlayDone = false;
          cPlayingStoppedTime = ::GetTickCount();
                        SessionNode *pSession = cSession;
                        if (!pSession)
                        {
                                debug("Session is gone. Cleaning up...");
                                cStoppingCompositePlay = false;
                                cCompositePlayStopped = true;
                                debug("Session is gone - Set cStoppingCompositePlay to false and
cCompositePlayStopped to true");
                                DMChannelState state = GetState();
                                if (state == S_PlayingFile)
                                {
                                        CloseDlgcFileHandle();
                                }
                                CleanupPlay();
                                SetState(S_Idle);
                    cPlayingStoppedTime = ::GetTickCount();
                        }
                        else
                        {
                                //min 9/12/2000 cStateLock.Enter();
                                //debug("TDX_PLAY - got the lock");
                                Synchronized sync(pSession);
                                DMChannelState state = GetState();
                                if (state == S_PlayingFile)
                                {
                                        CloseDlgcFileHandle();
                                }
                                switch(Reason)
                                {
```

```
                                        case TM_EOD:
                                        {
                                                debug("TDX_PLAY - CleanupPlay");
                                                CleanupPlay();
                                                debug("TDX_PLAY - CleanupPlay done");
                                                bNotifyPlayDone = true;
                                                break;
                                        }
                                        case TM_USRSTOP:
                                        {
                                                //
                                                // We don't notify if the Reason is TM_USRSTOP, since this
event
                                                // was initiated through the MSP, who will do the notification
                                                //
                                                if (cStoppingCompositePlay)
                                                {
                                                        cStoppingCompositePlay = false;
                                                        cCompositePlayStopped = true;
                                                        debug("TM_USRSTOP - Set cStoppingCompositePlay
to false and cCompositePlayStopped to true");
                                                }
                                                CleanupPlay();
                                                break;
                                        }
                                        default:
                                        {
                                                debug("reason for play stopped is neither TM_EOD nor
TM_USRSTOP");
                                                break;
                                        }
                                }
                                SetState(S_Idle);
                                cPlayingStoppedTime = ::GetTickCount();
                                sync.notifyAll();
                        }
        //cStateLock.Leave();
                        if (bNotifyPlayDone)
                        {
          debug("TDX_PLAY - calling NotifyPlayDone");
                                NotifyPlayDone();
          debug("TDX_PLAY - called NotifyPlayDone");
                        }
                        break;
                }
    case TDX_DIAL:
    {
                        Reason = ATDX_TERMMSK(chdesc);
                        debug("TDX_DIAL - DTMF playing is done, reason = %ld.", Reason);
                        bool bNotifyPlayDone = true;
      SessionNode *pSession = cSession;
      if (!pSession)
                        {
                                debug("Session is gone. Cleaning up...");
                                SetState(S_Idle);
                                cPlayingStoppedTime = ::GetTickCount();
```

```
                        }
                        else
                        {
                                Synchronized sync(pSession);
                                DMChannelState state = GetState();

                                if (Reason == TM_USRSTOP)
                                {
                                        // dx_dial cannot be stopped by using dx_stopch
                                        bNotifyPlayDone = false;
                                        warning("Magic! dx_dial cannot be stopped using dx_stopch");
                                }
                                SetState(S_Idle);
                                cPlayingStoppedTime = ::GetTickCount();
                                sync.notifyAll();
                        }
                        if (bNotifyPlayDone)
                        {
                                NotifyPlayDone();
                        }
          break;
        }
                case TDX_RECORD:
                {
          unsigned long recordingStoppedTime = ::GetTickCount();
                        Reason = ATDX_TERMMSK(chdesc);
                        debug("TDX_RECORD - record is done, reason = %ld.", Reason);
                        SessionNode *pSession = cSession;
                        if (!pSession)
                        {
                                debug("Session is gone. Cleaning up...");
                                DMChannelState state = GetState();
                                if (state == S_Recording)
                                {
                                        CloseDlgcFileHandle();
#ifdef USE_WFSTREAMER
                                        MediaChannel::DestroyStreamer();
#endif
                                }
                                SetState(S_Idle);
                                MediaChannel::SetState(MediaChannel::S_Idle);
                                cRecordingStoppedTime = recordingStoppedTime;
                        }
                        else
                        {
                                Synchronized sync(pSession);
                                DMChannelState state = GetState();
                                if (state == S_Recording)
                                {
                                        CloseDlgcFileHandle();
#ifdef USE_WFSTREAMER
                                        MediaChannel::DestroyStreamer();
#endif
                                }
                                SetState(S_Idle);
                                MediaChannel::SetState(MediaChannel::S_Idle);
```

```
                        cRecordingStoppedTime = recordingStoppedTime;
                        sync.notifyAll();
                }
                unsigned long recordingDuration = cRecordingStoppedTime - cRecordStartedTime;
                if (Reason == TM_MAXSIL)
                {
                        SessionNode *pSession = cSession;
                        if (pSession)
                        {
                                Synchronized sync(pSession);
                                if (!pSession->IsGoingAway())
                                {
                                        {
                                                RecordingDoneNotification::RecordingDoneReason
recDoneReason = RecordingDoneNotification::RDR_EndOfSpeech;
                                                // this can be bos or eos
                                                if (recordingDuration < (cBosTimeout + 150) &&
                                                        recordingDuration > (cBosTimeout - 150))
                                                        recDoneReason =
RecordingDoneNotification::RDR_NoSpeech;
                                                // notify client
                                                pSession->NotifyRecordingDone(cCurrentCallToken,
recDoneReason, recordingDuration);
                                                sync.notifyAll();
                                        }
                                }
                        }
                }
        else if (Reason == TM_USRSTOP)
                {
                        //
                        // We don't notify if the Reason is TM_USRSTOP, since this event
                        // was initiated through the MSP, who will do the notification
                        //
                }
                else
                        debug("MC %d got TDX_RECORD event, reason = %ld",
                    chdesc, ATDX_TERMMSK(chdesc));
                    break;
            }

            case TDX_ERROR:
            {
        SessionNode *pSession = cSession;
        if (!pSession)
                {
                        DMChannelState state = GetState();
                        debug("TDX_ERROR.");
                        if ((state == S_PlayingFile) ||
                                (state == S_Recording))
                        {
                                CloseDlgcFileHandle();
                        }
#ifdef USE_WFSTREAMER
                        MediaChannel::StopAndDestroyStreamer();
#endif
```

```
                                MediaChannel::NotifyFailure();
                                if (cStoppingCompositePlay)
                                {
                                        cStoppingCompositePlay = false;
                                        cCompositePlayStopped = true;
                                        debug("TDX_ERROR - Set cStoppingCompositePlay to false and
cCompositePlayStopped to true");
                                }
                                SetState(S_Idle);
                                cRecordingStoppedTime = ::GetTickCount();
                                cPlayingStoppedTime = ::GetTickCount();
                        }
                        else
                        {
         Synchronized sync(pSession);
                                DMChannelState state = GetState();
                                debug("TDX_ERROR.");
                                if ((state == S_PlayingFile) ||
                                        (state == S_Recording))
                                {
                                        CloseDlgcFileHandle();
                                }
        #ifdef USE_WFSTREAMER
                                MediaChannel::StopAndDestroyStreamer();
        #endif
                                MediaChannel::NotifyFailure();
                                switch (state)
                                {
                                        case S_PlayingFile:
                                        case S_PlayingStream:
                                        case S_PlayingTTS:
                                                {
                                                        SessionNode* s = cSession;
                                                        if (s != NULL) {
                                                                // GetMSP()->NotifyFailure(cClientID,
cCurrentCallToken, cMsgSerialNumber, MSPF_CANTPLAY);
                                                                s->NotifyMSPFailure(cCurrentCallToken,
cMsgSerialNumber, MSPF_CANTPLAY);
                                                        }
                                                }
                                                CleanupPlay();
                                                break;

                                        case S_Recording:
                                        case S_StreamRecording: {
                                                                                        SessionNode* s = cSession;
                                                                                        if (s != NULL) {
                                                                                                cSession-
>NotifyMSPFailure(cCurrentCallToken, cMsgSerialNumber, MSPF_CANTRECORD);
                                                                                        }
                                                                                        // GetMSP()-
>NotifyFailure(cClientID, cCurrentCallToken, cMsgSerialNumber, MSPF_CANTRECORD);
                                                                                                }
                                                break;
                                        default:
                                                break;
```

```
                                        }
                                        if (cStoppingCompositePlay)
                                        {
                                                cStoppingCompositePlay = false;
                                                cCompositePlayStopped = true;
                                                debug("TDX_ERROR - Set cStoppingCompositePlay to false and
cCompositePlayStopped to true");
                                        }
                                        SetState(S_Idle);
                                cRecordingStoppedTime = ::GetTickCount();
                                    cPlayingStoppedTime = ::GetTickCount();
                sync.notifyAll();
                }
                                error("got TDX_ERROR event, reason = %s, error id = %ld",
                        ATDV_ERRMSGP(chdesc), ATDV_LASTERR(chdesc));
                                break;
                        }
                        default:
                                warning("got unhandled event %ld", EventCode);
                                break;
                }
                return true;
}
unsigned long
DialogicMChannel::Play(const gm::string& filename, bool ttsSource, bool useCallback)
{
        debug("before waiting for TDX_PLAY termination event.");
   SessionNode *pSession = cSession;
   Synchronized sync(pSession);
   long timeLeft = 5000;
   long targetTime = timeLeft + ::GetTickCount();
   while (pSession->IsGoingAway() == false &&
       cPlayingStoppedTime == -1L && timeLeft > 0) {
      sync.wait(timeLeft);
      if (::GetTickCount() - targetTime > -1) {
         break;
      }
      timeLeft = targetTime - ::GetTickCount();
   }
   if (cPlayingStoppedTime == -1L)
        {
                error("cannot start playing because the previous playing is not ended yet.");
                return MSPF_CANTPLAY;
        }
   debug("after waiting for TDX_PLAY termination event.");
        if (useCallback)
        {
                dx_setchxfercnt(cHdl, 6); // Set dialogic to 1.5K buffer transfers for streaming to avoid playing too
long of a silence since samples may not be ready
                cIOTT.io_type = IO_DEV|IO_UIO|IO_EOT;
                cUIO.u_write  = DMChanPlayWrite;
                cUIO.u_read   = DMChanPlayRead;
                cUIO.u_seek   = DMChanPlaySeek;
                dx_setdevuio(cHdl, &cUIO, NULL);
        }
        else
```

```
        {
                dx_setchxfercnt(cHdl, 1); // Set dialogic to 8K buffer transfers
                cIOTT.io_type = IO_DEV | IO_EOT;
        }
        cIOTT.io_bufp = 0;
        cIOTT.io_offset = 0;
        cIOTT.io_length = -1;
        cIOTT.rfu = 0;
        if (useCallback)
        {
                cXPB.wFileFormat = FILE_FORMAT_VOX;
                cIOTT.io_fhandle = (int)this; // use this int to point to this object so it will be passed to the read
function
                if (!ttsSource)
                {
                        if (cpPlayFile)
                        {
                                fclose(cpPlayFile);
                                cpPlayFile = NULL;
                                debug("Closed previous cpPlayFile handle.");
                        }

                        // open the wave file for reading
                        if ((cpPlayFile = OpenFile((const char *)filename, "rb")) == NULL)
                        {
                                // cannot open wave/vox file
                                warning("Cannot open file \"%s\" for reading", (char *)(const char *)filename);
                                return MSPF_CANTFINDPROMPT;
                        }
                        else
                        {
                                debug("File \"%s\" opened for reading -- CurPos is set to 0.", (char *)(const char
*)filename);
                                cCurFilePos = 0;
                        }
                }
        }
        else
        {
                switch(CheckAudioFileFormat(filename))
                {
                        case (eAFF_Error):
                        {
                                return MSPF_CANTFINDPROMPT;
                        }
                        case (eAFF_Sphere):
                        {
                                cXPB.wFileFormat = FILE_FORMAT_VOX;
                                cIOTT.io_offset  = 1024;
                                break;
                        }
                        case (eAFF_CantDetermine): // right now "eAFF_CantDetermine" means wave file
                        default:
                        {
                                cXPB.wFileFormat = FILE_FORMAT_WAVE;
                        }
```

```
                }
                cIOTT.io_fhandle = dx_fileopen(filename, O_RDONLY|O_BINARY);
                if (cIOTT.io_fhandle == -1)
                        return MSPF_CANTFINDPROMPT;
        }
        dx_clrtpt(&cTPT, 1);
        cTPT.tp_type = IO_EOT;
        cTPT.tp_termno = DX_LCOFF;
        cTPT.tp_length = 0;
        cTPT.tp_flags = TF_LCOFF;
        cXPB.wDataFormat = DATA_FORMAT_MULAW;
        cXPB.nSamplesPerSec = DRT_8KHZ;
        cXPB.wBitsPerSample = 8;
  cPlayingStoppedTime = -1L;
        {
//              Synchronize sync(cStateLock);
                if (useCallback)
                        if (ttsSource)
                                SetState(S_PlayingTTS);
                        else
                                SetState(S_PlayingStream);
                else
                        SetState(S_PlayingFile);
    int adjustVolumeDB;
    if (adjustVolumeDB = ((DialogicMSP *)cServiceProvider)->GetDefaultAdjustPlayVolumeDB())
    {
      if (dx_adjsv(cHdl, SV_VOLUMETBL, SV_ABSPOS, adjustVolumeDB/2) == -1 )
      {
              error("dx_adjsv failed, reason = %s, error id = %ld",
                      ATDV_ERRMSGP(cHdl), ATDV_LASTERR(cHdl));
      }
    }
    if (dx_playiottdata(cHdl, &cIOTT, &cTPT, &cXPB, EV_ASYNC) == -1)
    {
      cPlayingStoppedTime = ::GetTickCount(); // set cPlayingStoppedTime here because we'll never get
TDX_PLAY to change the value
              error(" dx_playiottdata failed, reason = %s, error id = %ld",
                   ATDV_ERRMSGP(cHdl), ATDV_LASTERR(cHdl));
              // clean up for streaming
              if (ttsSource || useCallback)
                      CleanupPlay();
              else
                      CloseDlgcFileHandle();
              SetState(S_Idle);
              // end of cleanup
              return MSPF_CANTPLAY;
    }
        }
        debug("Playing");
        return MSPF_OK;
}
unsigned long
DialogicMChannel::PlaySilence(unsigned long durationmsecs)
{
        debug("in PlaySilence(), before waiting for TDX_PLAY termination event.");
  SessionNode *pSession = cSession;
```

```
Synchronized sync(pSession);
long timeLeft = 5000;
long targetTime = timeLeft + ::GetTickCount();
while (pSession->IsGoingAway() == false &&
    cPlayingStoppedTime == -1L && timeLeft > 0) {
  sync.wait(timeLeft);
  timeLeft = targetTime - ::GetTickCount();
}
if (cPlayingStoppedTime == -1L)
        {
                error("cannot start playing because the previous playing is not ended yet.");
                return MSPF_CANTPLAY;
        }
debug("in PlaySilence(), after waiting for TDX_PLAY termination event.");
        cSilenceBytes = durationmsecs * 8L;          // total number of bytes to be played
        // Set dialogic to 1.5 K buffer transfers
        //
        dx_setchxfercnt(cHdl, 6);
        // no file involved here
        cIOTT.io_type = IO_DEV|IO_UIO|IO_EOT;
        cUIO.u_write  = DMChanPlaySilenceWrite;
        cUIO.u_read   = DMChanPlaySilenceRead;
        cUIO.u_seek   = DMChanPlaySilenceSeek;
        dx_setdevuio(cHdl, &cUIO, NULL);
        cIOTT.io_bufp = 0;
        cIOTT.io_offset = 0;
        cIOTT.io_length = -1;
        cIOTT.rfu = 0;
        cXPB.wFileFormat = FILE_FORMAT_VOX;
        cIOTT.io_fhandle = (int)this; // use this int to point to this object so it will be passed to the read function
        dx_clrtpt(&cTPT, 1);
        cTPT.tp_type = IO_EOT;
        cTPT.tp_termno = DX_LCOFF;
        cTPT.tp_length = 0;
        cTPT.tp_flags = TF_LCOFF;
        cXPB.wDataFormat = DATA_FORMAT_MULAW;
        cXPB.nSamplesPerSec = DRT_8KHZ;
        cXPB.wBitsPerSample = 8;
        // Set Timer
/*      LARGE_INTEGER dueTime;
        dueTime.QuadPart = -((LONGLONG)(durationmsecs * (LONGLONG)10000));
        ::SetWaitableTimer(cSilencePromptTimer, &dueTime, 0L, NULL, 0, FALSE); // if this is not accurate
enough, handle it in the thread function itself
        ::ResetEvent(cStopTimerEvent);  // important -- do not move this to the thread function because if
dx_playiottdata fails, we need to set it (we need to guarantee the sequence)
        cSilencePromptTimerThreadH = (HANDLE)_beginthreadex( NULL,
                0,
                timerThreadFunc,
                (LPVOID) this,
                0,
                &cSilencePromptTimerThreadId);
        if (cSilencePromptTimerThreadH)
        {
                debug::printf("DialogicMChannel", "Silence Prompt Timer Thread launched\n");
        }
        else
```

```
        {
                debug::printf("DialogicMChannel", "Couldn't launch Silence Prompt Timer Thread\n");
                return MSPF_CANTPLAY;
        }
*/ //tmt we don't use timer anymore
  cPlayingStoppedTime = -1L;
        //::ResetEvent(cPlayTerminationEvent);
  {
//      Synchronize sync(cStateLock);
          SetState(S_PlayingSilence);
          if (dx_playiottdata(cHdl, &cIOTT, &cTPT, &cXPB, EV_ASYNC) == -1)
          {
       cPlayingStoppedTime = ::GetTickCount(); // set cPlayingStoppedTime here because we'll never get
TDX_PLAY to change the value
                SetState(S_Idle);
                error("dx_playiottdata failed in PlaySilence, reason = %s, error id = %ld",
              ATDV_ERRMSGP(cHdl), ATDV_LASTERR(cHdl));
                //dx_fileclose(cIOTT.io_fhandle);
                // clean up -- stop timer thread
                debug("dx_playiottdata failed in PlaySilence-- tell Silence Prompt Thread to stop.");
                ::SetEvent(cStopTimerEvent);
                // end of cleanup
                return MSPF_CANTPLAY;
          }
  }
        debug("Playing Silence");
        return MSPF_OK;
}
unsigned long
DialogicMChannel::PlayDTMF(const gm::string& dtmfString)
{
        debug("before waiting for play termination event.");
  SessionNode *pSession = cSession;
  Synchronized sync(pSession);
  long timeLeft = 5000;
  long targetTime = timeLeft + ::GetTickCount();
  while (pSession->IsGoingAway() == false &&
      cPlayingStoppedTime == -1L && timeLeft > 0) {
    sync.wait(timeLeft);
    timeLeft = targetTime - ::GetTickCount();
  }
  if (cPlayingStoppedTime == -1L)
        {
                error("cannot start playing because the previous playing is not ended yet.");
                return MSPF_CANTPLAY;
        }
        debug("after waiting for play termination event.");
  cPlayingStoppedTime = -1L;
        {
//              Synchronize sync(cStateLock);
    SetState(S_PlayingDTMF);
    if (dx_dial(cHdl, dtmfString.c_str(), NULL, EV_ASYNC) == -1)
    {
      // failure
      cPlayingStoppedTime = ::GetTickCount(); // set cPlayingStoppedTime here because we'll never get
TDX_PLAY to change the value
```

```
                SetState(S_Idle);
                error("dx_dial failed in PlayDTMF, reason = %s, error id = %ld",
            ATDV_ERRMSGP(cHdl), ATDV_LASTERR(cHdl));
        return MSPF_CANTPLAY;
      }
      else
      {
        debug("Playing dtmf");
      }
    }
    return MSPF_OK;
}
unsigned long
DialogicMChannel::Stop(void)
{
        //if (dx_stopch(cHdl, EV_SYNC) == -1)
#ifdef ENABLE_PERF_MEASURE
        cDoStopTime = ::GetTickCount();
#endif
    //cRecordingStoppedTime = -1L;
        unsigned long tnow = ::GetTickCount();
        if (tnow >= cRecordStartedTime)
        {
                unsigned long timeElapsedSinceRecordingStarted = tnow - cRecordStartedTime;
                diag("timeElapsedSinceRecordingStarted is %ld.", timeElapsedSinceRecordingStarted);
                if (timeElapsedSinceRecordingStarted < (unsigned long)500)
                {
                        diag("sleep %ld msecs.", 500-timeElapsedSinceRecordingStarted);
                        ::Sleep(500-timeElapsedSinceRecordingStarted);
                }
        }
    debug("calling dx_stopch()");
        if (dx_stopch(cHdl, EV_ASYNC) == -1)
    {
      debug("called dx_stopch() - failed");
                return MSPF_CANTSTOP;
    }
    else
      debug("called dx_stopch() - successful");
        return MSPF_OK;
}
unsigned long
DialogicMChannel::Record(const gm::string& filename, bool bStreaming, int bos_timeout, int tms_timeout, int
eos_timeout)
{
        debug("before waiting for TDX_RECORD termination event.");
    cBosTimeout = bos_timeout;
    cTmsTimeout = tms_timeout;
    cEosTimeout = eos_timeout;
    SessionNode *pSession = cSession;
    Synchronized sync(pSession);
    long timeLeft = 5000;
    long targetTime = timeLeft + ::GetTickCount();
    while (pSession->IsGoingAway() == false &&
        cRecordingStoppedTime == -1L && timeLeft > 0) {
      sync.wait(timeLeft);
```

```
    timeLeft = targetTime - ::GetTickCount();
  }
        if (cRecordingStoppedTime == -1L || pSession->IsGoingAway())
        {
                if (pSession->IsGoingAway())
                        info("cannot start recording because session is going away");
                else
                        error("cannot start recording because the previous recording is not ended yet.");
                if (!bStreaming)
                        _unlink(filename);
                return MSPF_CANTRECORD;
        }
  //
  // now allow dialogic to settle before asking to record
  targetTime = cRecordingStoppedTime + 200L;
  timeLeft = targetTime - ::GetTickCount();
  while (pSession->IsGoingAway() == false &&
      cRecordingStoppedTime && timeLeft > 0)
  {
    debug("Waiting %ld msecs before asking Dialogic to start recording", timeLeft);
    sync.wait(timeLeft);
    timeLeft = targetTime - ::GetTickCount();
  }
  if (pSession->IsGoingAway()) {
    if (!bStreaming)
      _unlink(filename);
    return MSPF_CANTRECORD;
  }
        debug("after waiting for TDX_RECORD termination event.");
        // Set dialogic to 1.5K buffer transfers, for
        // streaming recording
        //
        dx_setchxfercnt(cHdl, 6);
        if (bStreaming)
        {
                cIOTT.io_type = IO_DEV|IO_UIO|IO_EOT;
                cUIO.u_write  = DMChanRecordWrite;
                cUIO.u_read   = DMChanRecordRead;
                cUIO.u_seek   = DMChanRecordSeek;
                dx_setdevuio(cHdl, &cUIO, NULL);
        }
        else
                cIOTT.io_type = IO_DEV | IO_EOT;
        cIOTT.io_bufp = 0;
        cIOTT.io_offset = 0;
        cIOTT.io_length = -1;
        cIOTT.rfu = 0;
        if (bStreaming)
        {
                cIOTT.io_fhandle = (int)this; // use this int to point to this object so it will be passed to the write
function
        }
        else
        {
                debug("before opening file.");
                cIOTT.io_fhandle = dx_fileopen(filename,
```

```
                        _O_RDWR | _O_CREAT | _O_TRUNC | _O_BINARY, _S_IREAD | _S_IWRITE);
                debug("after opening file.");
    if (cIOTT.io_fhandle == -1) {
                        error("invalid file handle.");
                        return MSPF_CANTFINDPROMPT;
                }
        }
    DV_TPT tpt;
        dx_clrtpt(&tpt, 1);
    if (bos_timeout < 0 || eos_timeout < 0)
    {
            tpt.tp_type = IO_EOT;
            tpt.tp_termno = DX_LCOFF;
            tpt.tp_length = 0;
            tpt.tp_flags = TF_LCOFF;
    }
    else
    {
      tpt.tp_termno = DX_MAXSIL;
            tpt.tp_length = eos_timeout/10;
      tpt.tp_flags = TF_MAXSIL | TF_SETINIT | TF_10MS    ;
      tpt.tp_data = bos_timeout/10;
    }
        if (bStreaming)
                cXPB.wFileFormat = FILE_FORMAT_VOX;
        else
                cXPB.wFileFormat = FILE_FORMAT_WAVE;

        cXPB.wDataFormat = DATA_FORMAT_MULAW;
        cXPB.nSamplesPerSec = DRT_8KHZ;
        cXPB.wBitsPerSample = 8;
        debug("before calling dx_reciottdata.");
    {
//      Synchronize sync(cStateLock);
            if (bStreaming)
                    SetState(S_StreamRecording);
            else
                    SetState(S_Recording);
#ifdef ENABLE_PERF_MEASURE
            // get time for start recording
            cFirstCallback = true;
            cCallbackTime = ::GetTickCount();
            SetStartRecordingTime(cCallbackTime);
#endif
            //if (dx_reciottdata(cHdl, &cIOTT, &cTPT, &cXPB, MD_NOGAIN | EV_ASYNC) == -1)
            cRecordingStoppedTime = -1L;
            if (dx_reciottdata(cHdl, &cIOTT, &tpt, &cXPB, MD_NOGAIN | EV_ASYNC) == -1)
            {
        cRecordingStoppedTime = ::GetTickCount(); // set cRecordingStoppedTime here because we'll never get
TDX_RECORD to change the value
                    SetState(S_Idle);
                    debug("dx_reciottdata failed, reason = %s, error id = %ld",
                ATDV_ERRMSGP(cHdl), ATDV_LASTERR(cHdl));
                    if (!bStreaming) {
                            CloseDlgcFileHandle();
                            _unlink(filename);
```

```
                }
                return MSPF_CANTRECORD;
        }
                else
                {
                        cRecordStartedTime = ::GetTickCount();
                }
    }
    debug("Recording");
        return MSPF_OK;
}
// virtual
unsigned long
DialogicMChannel::ListenTo(unsigned short timeslot)
{
        if (timeslot != (unsigned short)-1)
        {
                SC_TSINFO tsinfo;
                long timeslots = timeslot;
                debug(" listen to timeslot %d", timeslot);

                tsinfo.sc_numts = 1;
                tsinfo.sc_tsarrayp = &timeslots;
                if (dx_listen(cHdl, &tsinfo) == -1)
                {
                        error("dx_listen failed!");
                        return TSPF_CANTLISTEN;
                }
        }
        else
        {
                debug("Unlistening");
                if (dx_unlisten(cHdl) == -1)
                {
                        error("dx_unlisten failed!");
                        return TSPF_CANTLISTEN;
                }
        }
        return TSPF_OK;
}
int
DialogicMChannel::DMChanRecordWrite(int fHandle, char *buffer, unsigned int count)
{
        unsigned long callbacktime = ::GetTickCount();
    if (!fHandle) {
        DLGCMSPLog::debug("time between callbacks: %ld ", callbacktime - cCallbackTime);
        cCallbackTime = callbacktime;
      DLGCMSPLog::error("DIALOGIC ERROR, got null pointer in DMChanRecordWrite from thread %ld",
                GetCurrentThreadId());
                  return 0;
        }

    DialogicMChannel* pMChannel = (DialogicMChannel*)fHandle;
    pMChannel->debug("DMChanRecordWrite: thread=%ld, time between callbacks: %ld",
                GetCurrentThreadId(),
                callbacktime - cCallbackTime);
```

```
    cCallbackTime = callbacktime;
            // send to listener
            if (cFirstCallback)
            {
                    pMChannel->debug("Media Channel %p, time it took to get the first recording callback: %ld
msecs.", (DialogicMChannel*)fHandle, callbacktime - ((DialogicMChannel *)fHandle)->GetStartRecordingTime());
                    cFirstCallback = false;
            }
            SampleListener *pSListener = pMChannel->GetSampleListener();
    if (pSListener) {
      ((DialogicMSP*)(pMChannel->GetMSP()))->sendSamplingRequest(pSListener,
                                            (unsigned char*) buffer,
                                            (unsigned int)count);
      pMChannel->debug("fed sample");
    }
//                          pSListener->NotifySample((const unsigned char *)buffer, count);
            return count;
}
// dummy read function
int
DialogicMChannel::DMChanRecordRead(int fHandle, char *buffer, unsigned int count)
{
            return 1;
}
// dummy seek function
long
DialogicMChannel::DMChanRecordSeek(int fHandle, long offset, int initPos)
{
            return offset;
}
int
DialogicMChannel::DMChanPlayWrite(int fHandle, char *buffer, unsigned int count)
{
            return 1;
}
// dummy read function
int
DialogicMChannel::DMChanPlayRead(int fHandle, char *buffer, unsigned int count)
{
            if (!fHandle)
    {
            DLGCMSPLog::error("DIALOGIC ERROR, got null pointer in DMChanPlayRead");
      return 0;
    }
    DialogicMChannel *pChan = (DialogicMChannel *)fHandle;
            if (pChan->GetState() == S_PlayingStream)
            {
                    // read from file and fill in the buffer
                    return pChan->StreamRead(buffer, count);
            }
            else
            {
      SessionNode *pSession = ((DialogicMChannel *)fHandle)->cSession;
                if (!pChan->VerifySession(pSession))
                        return 0;
```

```
        pChan->debug("DMChanPlayRead: trying to get lock on session");
    Synchronized sync(pSession);
        pChan->debug("DMChanPlayRead: got lock on session");
    if (pSession->IsGoingAway())
      return 0;
    pChan->debug("DMChanPlayRead: Get next samples");
            return pSession->cTTSChannel->GetNextSamples(buffer, count);
        }
}
// dummy seek function
long
DialogicMChannel::DMChanPlaySeek(int fHandle, long offset, int initPos)
{
        //debug::printf("DialogicMChannel","offset = %ld, initPos = %d\n", offset, initPos);
        return offset;
}
int
DialogicMChannel::StreamRead(char *buf, unsigned int count)
{
        size_t numCharsRead = 0;
        long eofPos = 0L;
        if (::WaitForSingleObject(cStreamPlayReadMutex, 0) != WAIT_TIMEOUT) {

            if (cpPlayFile) {
                fseek(cpPlayFile, 0, SEEK_END);
                eofPos = ftell(cpPlayFile);
                debug("StreamRead:  Cur Pos = %d,  EOF Pos = %d", cCurFilePos, eofPos);
                if (::WaitForSingleObject(cEndOfStreamPlayDataEvent, 0) == WAIT_TIMEOUT) {
                    // see how many bytes we got
                    //
                    if ((eofPos - cCurFilePos) < count) {
                        // we got less than asked
                        // return count and silence
                        numCharsRead = count;
                        memset(buf, 0xFF, count); // 'FF' is silence
                        debug("StreamRead:  not EOD -- only %d characters available for reading
-- returning silence", eofPos - cCurFilePos);
                    }
                    else {
                        fseek(cpPlayFile, cCurFilePos, SEEK_SET); // back to current pos
                        numCharsRead = fread((char *)buf, sizeof(char), count, cpPlayFile);
                        cCurFilePos = ftell(cpPlayFile); // advance file ptr
                        debug("StreamRead:  not EOD -- Read %d characters", numCharsRead);
                    }
                }
                else {
                    // end of data
                    // only stop if eof is reached
                    if (eofPos == cCurFilePos) {
                        ::ResetEvent(cEndOfStreamPlayDataEvent);
                        debug("StreamRead:  EOD EOF -- Reset EndOfDataEvent");
                        numCharsRead = 0;
                        debug("StreamRead:  EOD EOF -- Reset End of Data event");
                    }
                    else {
                        fseek(cpPlayFile, cCurFilePos, SEEK_SET); // back to current pos
```

```
                                        numCharsRead = fread((char *)buf, sizeof(char), count, cpPlayFile);
                                        cCurFilePos = ftell(cpPlayFile); // advance file ptr
                                        if (numCharsRead < count) {
                                                ::ResetEvent(cEndOfStreamPlayDataEvent);
                                                debug("StreamRead:  EOD, not EOF -- Reset EndOfDataEvent");
                                        }
                                        debug("StreamRead:  EOD, not EOF -- Read %d characters",
numCharsRead);
                                }
                        }
                }
                debug("StreamRead:  Num of Chars Read: %d ", numCharsRead);
                ::ReleaseMutex(cStreamPlayReadMutex);
        }
        return numCharsRead;
}
FILE *
DialogicMChannel::OpenFile(const char *filename, const char *mode)
{
        FILE *pFile = NULL;
        if (!filename ||
                ((pFile = fopen(filename, mode)) == NULL)) {
                warning("DialogicMChannel", "Cannot open file %s", filename);
        }
        else
                debug("File %s opened successfully", filename);
        return pFile;
}
DialogicMChannel::AudioFileFormat
DialogicMChannel::CheckAudioFileFormat(const gm::string& filename)
{
        AudioFileFormat ret = eAFF_CantDetermine;
        const size_t signatureSize = 7;
        char fSignatureBuf[signatureSize+1];
        FILE *pFile = OpenFile(filename, "rb");
        if (pFile == NULL) {
                ret = eAFF_Error;
        }
        else {
                fseek(pFile, 0L, SEEK_SET);
                size_t numCharsRead = fread(fSignatureBuf, sizeof(char), signatureSize, pFile);
                fSignatureBuf[signatureSize] = '\0';
                fclose(pFile);
                if ((numCharsRead == signatureSize) &&
                        !strcmp(fSignatureBuf,"NIST_1A")) {
                        ret = eAFF_Sphere;
                        debug("\"%s\" is a sphere wave file", (char *)(const char *)filename);
                }
                else {
                        ret = eAFF_CantDetermine;
                        warning("\"%s\" -- cannot determine audio format - assuming wave format for now", (char
*)(const char *)filename);
                }
        }
        return ret;
}
```

```
// dummy write function
int
DialogicMChannel::DMChanPlaySilenceWrite(int fHandle, char *buffer, unsigned int count)
{
        return 1;
}
// dummy read function
int
DialogicMChannel::DMChanPlaySilenceRead(int fHandle, char *buffer, unsigned int count)
{
        count = ((DialogicMChannel *)fHandle)->NumSilenceBytesToPlay(count);
        memset(buffer, 0xFF, count); // 'FF' is silence
   DLGCMSPLog::debug("DMChanPlaySilenceRead:  returning silence -- %d samples, Media Channel %p.", count,
(DialogicMChannel *)fHandle);
        return count;
}
unsigned int
DialogicMChannel::NumSilenceBytesToPlay(unsigned int count)
{
        if (cSilenceBytes < count)
        {
                count = cSilenceBytes;
                cSilenceBytes = 0L;
        }
        else
                cSilenceBytes -= count;
        return count;
}
// dummy seek function
long
DialogicMChannel::DMChanPlaySilenceSeek(int fHandle, long offset, int initPos)
{
        //debug::printf("DialogicMChannel","offset = %ld, initPos = %d\n", offset, initPos);
        return offset;
}
unsigned __stdcall
DialogicMChannel::timerThreadFunc(LPVOID param)
{
        DialogicMChannel *pThisObj = (DialogicMChannel *)param;
        return pThisObj->TimerThreadFunc();
}
unsigned
DialogicMChannel::TimerThreadFunc(void)
{
        while (true) {
                if (::WaitForSingleObject(cStopTimerEvent, 0) != WAIT_TIMEOUT) {
                        // signaled, exit thread
                        debug("Silence Prompt Timer Thread is requested to stop...");
                        ::ResetEvent(cStopTimerEvent);
                        break;
                }
                if (::WaitForSingleObject(cSilencePromptTimer, 3) != WAIT_TIMEOUT) {
                        // signaled
                        // stop the silence prompt play and exit thread
                        debug("stopping silence prompt.");
                        Stop();
```

```
                        debug("stopped silence prompt - exiting thread...");
                        break;
                }
        }
        ::CloseHandle(cSilencePromptTimerThreadH);
        cSilencePromptTimerThreadH = 0;
        return 0;
}
void
DialogicMChannel::CleanupPlay(void)
{
        switch (GetState())
        {
        case S_PlayingStream:
                if (cpPlayFile)
                {
                        if (::WaitForSingleObject(cStreamPlayReadMutex, 500) != WAIT_TIMEOUT) {
                                debug("Closing cpPlayFile (Stream Play File) ...");
                                fclose(cpPlayFile);
                                cpPlayFile = NULL;
                                ::ReleaseMutex(cStreamPlayReadMutex);
                        }
                        else
                        {
                                error("Error: cpPlayFile (Stream Play File) cannot be closed.");
                        }
                        break;
        case S_PlayingTTS:
                StopTTS();      // Is this needed????
                break;
        default:
                break;
                }
        }
}
void
DialogicMChannel::CloseDlgcFileHandle(void)
{
        int dxRet = dx_fileclose(cIOTT.io_fhandle);
        debug("dx_fileclose() returns %d.", dxRet);
        cIOTT.io_fhandle = 0;
}
```

```
/*
 * DialogicMSP.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.20 $
 * $Date: 2000/12/07 03:25:08 $
 */
static const char* id="$Id: DialogicMSP.cpp,v 1.20 2000/12/07 03:25:08 Mitsuru Oshima Exp $";
#include <windows.h>
#include <stdio.h>
#include "DialogicMSP.h"
#include "DialogicMChannel.h"
#include "MTSLib/MTSServer.h"
#include "MTSLib/PromptSegment.h"
#include "MTSLib/RangeParser.h"
#include "NotifySampleEvent.h"
#include "VPPUtils/Thread.h"
#include "VPPUtils/Synchronized.h"
#include "VPPUtils/ConfigMgr.h"
DialogicMSP *DialogicMSP::cInstance = 0;
const char const *DialogicMSP::cLogType="DLGC_MSP";
//int DialogicMSP::cDefaultAdjustPlayVolumeDB = 0; // 0 DB - no speed adjustment
DialogicMSP::DialogicMSP(MTSServer *pServer)
        : MediaServiceProvider(pServer, cLogType), samplingQueue(), queueState(QUEUE_RUNNING),
numThreads(4)
{
        GetMTSServer()->GetSystemProperties().SetProperty("Media.ReleaseString",
                "Dialogic VOX Device MSP Release 0.1");
  long value;
  // set default for volume adjustment for play
  if (GetMTSServer()->GetSessionMgr()->GetDefaultProperty("Media.DefaultAdjustPlayVolumeDB", value))
  {
    cDefaultAdjustPlayVolumeDB = value;
    if (cDefaultAdjustPlayVolumeDB < -8)
      cDefaultAdjustPlayVolumeDB = -8;
    else if (cDefaultAdjustPlayVolumeDB > 8)
      cDefaultAdjustPlayVolumeDB = 8;
    else
    {
      if (cDefaultAdjustPlayVolumeDB % 2)
        cDefaultAdjustPlayVolumeDB = cDefaultAdjustPlayVolumeDB < 0 ? cDefaultAdjustPlayVolumeDB - 1 :
cDefaultAdjustPlayVolumeDB + 1;
    }
  }
  else
    cDefaultAdjustPlayVolumeDB = 0;
  for (int i=0; i<numThreads; i++) {
    (new ThreadT<DialogicMSP>(this, DialogicMSP::samplingMainLoop))->start();
  }
}
// virtual
```

```
DialogicMSP::~DialogicMSP(void)
{
  Synchronized sync(samplingQueue);
  queueState = QUEUE_STOPPING;
  while ( samplingQueue.peek() != NULL) {
    delete samplingQueue.pop();
  }
  samplingQueue.stop();
  while (queueState == QUEUE_STOPPING) {
    sync.wait();
  }
}
void
DialogicMSP::sendSamplingRequest(SampleListener* pSListener,
                    unsigned char* buffer,
                    unsigned int count) {
  samplingQueue.push(new NotifySampleEvent(pSListener, buffer, count));
}
void
DialogicMSP::samplingMainLoop() {
  while (queueState == QUEUE_RUNNING) {
    NotifySampleEvent* ev = samplingQueue.pop();
    if (ev != NULL) {
      ev->notifySample();
      delete ev;
    }
  }
  {
    Synchronized sync(samplingQueue);
    numThreads --;
    if (numThreads == 0) {
      queueState = QUEUE_STOPPED;
      sync.notify();
    }
  }
}
bool
DialogicMSP::Setup(void)
{
        int boardcnt = 0;
        sr_getboardcnt(DEV_CLASS_VOICE, &boardcnt);
        bool bAllocateAll = false;
        gm::string str;
        gm::string mspname;
        gm::string parent(MTS_REGISTRY_BASE);
        gm::string newparent;
        if (!ConfigMgr::GetString(parent, "Media Service Provider", mspname))
                bAllocateAll = true;
        else
        {
                newparent = parent;
                newparent += ".Media Service Providers.";
                newparent += mspname;
        }
        if (!bAllocateAll && !ConfigMgr::GetString(newparent, "Channels", str))
                bAllocateAll = true;
```

```
        RangeParser range(str);
  int num = 0;
        for (int i = 0; i < boardcnt; i++)
        {
                char devname[64];
                // Create channels
                //
                sprintf(devname, "dxxxB%d", i + 1);
                int bddev = dx_open(devname, 0);
                if (bddev == -1)
                        return false;
                long subdevs = ATDV_SUBDEVS(bddev);
                char **chnamesp = ATDX_CHNAMES(bddev);
                // Allocate channel objects for this board
                //
                if (!bAllocateAll)
                {
                        for (int j = 0; j < (int)subdevs; j++)
                        {
                                if (range.IsSet((i*subdevs)+j))
                                {
                                        NewChannel(chnamesp[j], (i*subdevs)+j);
                                }
                        }
                }
                else
                {
                        for (int j = 0; j < (int)subdevs; j++)
                        {
                                NewChannel(chnamesp[j], (i*subdevs)+j);
                        }
                }
                dx_close(bddev);
        }
        return true;
}
bool
DialogicMSP::NewChannel(const char *name, int chanNum)
{
        bool ret = false;
        DialogicMChannel *pChannel = new DialogicMChannel(this, cLogType, chanNum);
        if (pChannel && pChannel->Setup(name))
        {
                AddChannel(pChannel);
                ret = true;
        }
        else
        {
                delete pChannel;
                error("Error : Dialogic Media Channel Setup failed for device \"%s\".", name);
        }
        return ret;
}
// virtual
bool
DialogicMSP::Teardown(void)
```

```
{
        // Channels are deleted by superclass
        //
        return MediaServiceProvider::Teardown();
}
DialogicMChannel *
DialogicMSP::FindChannelForDescriptor(int desc)
{
        std::list<Channel *>::iterator iter = cChannels.begin();
        while ((iter != cChannels.end()) &&
                ((DialogicMChannel *)(*iter))->GetHdl() != desc)
                iter++;
        return (iter != cChannels.end()) ? (DialogicMChannel *)(*iter) : 0;
}
DialogicMSP *
DialogicMSP::GetInstance(void)
{
        return cInstance;
}
void
DialogicMSP::SetInstance(DialogicMSP *ptr)
{
        cInstance = ptr;
}
MediaServiceProvider *
CreateMediaServiceProvider(MTSServer *pServer)
{
        DialogicMSP::SetInstance(new DialogicMSP(pServer));
        return DialogicMSP::GetInstance();
}
```

```
/*
 * DropCallMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:47 $
 */
static const char* id="$Id: DropCallMsg.cpp,v 1.2 2000/12/07 03:21:47 Mitsuru Oshima Exp $";
#include "DropCallMsg.h"
// virtual
void
DropCallMessage::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        cCDRInfo.Serialize(this);
}
// virtual
void
DropCallMessage::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        cCDRInfo.Deserialize(this);
}
```

```
/*
 * DTMFNtfy.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:47 $
 */
static const char* id="$Id: DTMFNtfy.cpp,v 1.2 2000/12/07 03:21:47 Mitsuru Oshima Exp $";
#include "DTMFNtfy.h"
// virtual
void
DTMFNotification::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        InsertUchar(cDTMF);
        InsertUlong(cTimeStamp);
}
// virtual
void
DTMFNotification::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        cDTMF = (char)ExtractUchar();
        cTimeStamp = (unsigned long)ExtractUlong();
}
```

```
/*
 * DTMFSegment.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:19:47 $
 */
static const char* id="$Id: DTMFSegment.cpp,v 1.2 2000/12/07 03:19:47 Mitsuru Oshima Exp $";
#include "DTMFSegment.h"
#include "MediaChannel.h"
// virtual
DTMFSegment::~DTMFSegment(void)
{
}
// virtual
bool
DTMFSegment::Play(MediaChannel *pMediaChannel,
                  CallToken& ct)
{
        return (bool)(pMediaChannel->DoPlayDTMF(ct, cDTMF) == MSPF_OK);
}
```

```
/*
 * DynamicGrammarMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:47 $
 */
static const char* id="$Id: DynamicGrammarMsg.cpp,v 1.2 2000/12/07 03:21:47 Mitsuru Oshima Exp $";
#include "DynamicGrammarMsg.h"
// virtual
void
DynamicGrammarMessage::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        InsertString(cDBKey);
}
// virtual
void
DynamicGrammarMessage::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        ExtractString(cDBKey);
}
```

```
/*
 * DynamicGrammarPhraseMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:48 $
 */
static const char* id="$Id: DynamicGrammarPhraseMsg.cpp,v 1.2 2000/12/07 03:21:48 Mitsuru Oshima Exp $";
#include "DynamicGrammarPhraseMsg.h"
// virtual
void
DynamicGrammarPhraseMessage::Serialize(void)
{
        DynamicGrammarMessage::Serialize();
        InsertString(cPhraseID);
}
// virtual
void
DynamicGrammarPhraseMessage::Deserialize(void)
{
        DynamicGrammarMessage::Deserialize();
        ExtractString(cPhraseID);
}
```

```
/*
 * DynamicGrammarPhraseResultNtfy.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:48 $
 */
static const char* id="$Id: DynamicGrammarPhraseResultNtfy.cpp,v 1.2 2000/12/07 03:21:48 Mitsuru Oshima Exp $";
#include "DynamicGrammarPhraseResultNtfy.h"
// virtual
void
DynamicGrammarPhraseResultNotification::Serialize(void)
{
        DynamicGrammarResultNotification::Serialize();
        InsertString(cPhraseID);
}
// virtual
void
DynamicGrammarPhraseResultNotification::Deserialize(void)
{
        DynamicGrammarResultNotification::Deserialize();
        ExtractString(cPhraseID);
}
```

```
/*
 * DynamicGrammarResultNtfy.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:48 $
 */
static const char* id="$Id: DynamicGrammarResultNtfy.cpp,v 1.2 2000/12/07 03:21:48 Mitsuru Oshima Exp $";
#include "DynamicGrammarResultNtfy.h"
// virtual
void
DynamicGrammarResultNotification::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        InsertString(cDBKey);
        InsertUshort((unsigned short)cResult);
}
// virtual
void
DynamicGrammarResultNotification::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        ExtractString(cDBKey);
        cResult = (Result)ExtractUshort();
}
```

```
/*
 * FailureNtfy.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:21:48 $
 */
static const char* id="$Id: FailureNtfy.cpp,v 1.5 2000/12/07 03:21:48 Mitsuru Oshima Exp $";
#include "FailureNtfy.h"
// virtual
void
FailureNotification::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        InsertUshort(cFailureSerialNo);
        InsertUlong(cInfo);
}
// virtual
void
FailureNotification::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        cFailureSerialNo = (SerialNumberT)ExtractUshort();
        cInfo = ExtractUlong();
}
```

```
#include "GlobalCallTChannel.h"
#include <windows.h>
#include <stdio.h>
#include <iostream.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <dtilib.h>
#include <gcisdn.h>
#include <gcerr.h>
#include "VPPMsg/VPPDefs.h"
#include "VPPUtils/VPPDebug.h"
#include "GlobalCallTSP.h"
#include "VPPMsg/Address.h"
#include "VPPUtils/Synchronized.h"
#include "MTSLib/SessionNode.h"
int GlobalCallTChannel::cSequenceNumber = 1;
CriticalSection GlobalCallTChannel::cCriticalSection;
static const unsigned long WAIT_TIME_BEFORE_DIALING = 1000L;
char const *GlobalCallTChannel::cStates[9] =
{
        "Null State|",
        "Disconnected State|",
        "Idle State|",
        "Offered State|",
        "Accepted State|",
        "Dialing State|",
        "Connected State|",
        "Alerting State|",
        "Undefined State|",
};
GlobalCallTChannel::GlobalCallTChannel(ServiceProvider *pServiceProvider, int boardNum, int portNum, const char
*logType, int id)
        : TelephonyChannel(pServiceProvider, logType, id, true),
          cBoardNum(boardNum), cPortNum(portNum),
          cHdl(-1), cWaitingForCall(false),
          cReady(false), cFlags(0), cMyCallState(eNullState),
          cNotifyCallDropped(false), cTimeChannelReleased(0L)
{
        //cInCallConnectedEvt = ::CreateEvent(NULL, TRUE, FALSE, NULL);
        //cWaitingForTerminationAckEvt = ::CreateEvent(NULL, TRUE, FALSE, NULL);
}
// virtual
GlobalCallTChannel::~GlobalCallTChannel(void)
{
        //::CloseHandle(cInCallConnectedEvt);
        //::CloseHandle(cWaitingForTerminationAckEvt);
}
bool
GlobalCallTChannel::Shutdown(void)
{
        int             err = GC_SUCCESS;
        int             libId;
        long    libErr;
        char    *pErrMsg;
        if ((err = gc_Close(cHdl)) == GC_SUCCESS) {
                info("%s GlobalCallTChannel shutdown finished.", GetCallState());
```

```
                return true;
        }
        else {
                error("%s GlobalCallTChannel shutdown failed.", GetCallState());
                if (gc_ErrorValue(&err, &libId, &libErr) != GC_SUCCESS) {
                        error("%s Could not get error value -- gc_ErrorValue failed", GetCallState());
                }
                else{
                        if (gc_ResultMsg( LIBID_GC, (long)err, &pErrMsg ) != GC_SUCCESS) {
                                error("%s Could not get error message -- gc_ResultMsg failed", GetCallState());
                        }
                        else {
                                error("%s GlobalCallTChannel shutdown error message: 0x%lx, %s",
GetCallState(), (long)err, pErrMsg);
                        }
                }
                return false;
        }
}
bool
GlobalCallTChannel::Setup(const gm::string& chname)
{
        cMyCallState = eNullState;
        if (gc_Open(&cHdl, (char *)(const char *)chname, 0) == GC_SUCCESS) {
/*
                int ret;
                ret = sr_enbhdlr(cHdl, EV_ANYEVT, dt_handler);
                if (ret == -1) {
                        cout << " sr_enbhdlr failed -- 0x%x " << ATDV_LASTERR(cHdl) << endl;
                }
                else {
                        cout << " sr_enbhdlr succeeded. " << endl;
                }
*/
                //((GlobalCallTSP *)GetTSP())->MonitorDevice(cHdl);
                SetIsISDN(true);
        }
        else {
                error("%s Channel setup error - gc_Open() failed.", GetCallState());
                return false;
        }
        long ts;
        SC_TSINFO tsinfo;
        tsinfo.sc_numts = 1;
        tsinfo.sc_tsarrayp = &ts;
        if (dt_getxmitslot(cHdl, &tsinfo) == -1) {
                error("%s Channel setup error: dt_getxmitslot failed!", GetCallState());
                return false;
        }
        diag("%s Telephony channel %p, %s is transmitting on timeslot %d",
                GetCallState(), this, ATDV_NAMEP(cHdl), (unsigned short)ts);
        cTxTimeslot = (unsigned short)ts;
//      int ret = gc_SetEvtMsk(cHdl,
GCMSK_ALERTING|GCMSK_BLOCKED|GCMSK_UNBLOCKED|GCMSK_PROCEEDING|GCMSK_PROC_SE
ND|GCMSK_PROGRESS|GCMSK_SETUP_ACK, GCACT_ADDMSK);
/*      int ret = gc_SetEvtMsk(cHdl, GCMSK_BLOCKED|GCMSK_UNBLOCKED, GCACT_ADDMSK);
```

```
        if (ret != GC_SUCCESS) {
                printMessage("GlobalCallTChannel", "Error: gc_SetEvtMsk() failed!");
                return false;
        }
*/
        if (gc_ResetLineDev(cHdl, EV_ASYNC) != GC_SUCCESS) {  // disconnect any active calls on the line
device
                return false;
        }
        return (bool)(cHdl != -1);
}
bool
GlobalCallTChannel::EnableHandler(void)
{
        int ret;
        ret = sr_enbhdlr(EV_ANYDEV, GCEV_UNBLOCKED, dt_handler);
        //ret = sr_enbhdlr(EV_ANYDEV, EV_ANYEVT, dt_handler);
        if (ret == -1) {
                //cout << " sr_enbhdlr failed -- 0x%x " << ATDV_LASTERR(cHdl) << endl;
                GCTSPLog::error("sr_enbhdlr failed.");
                return false;
        }
        else {
                GCTSPLog::diag("GlobalCallTChannel", "sr_enbhdlr succeeded.");
        }
/*      ret = gc_SetEvtMsk(cHdl,  GCMSK_ALERTING     | GCMSK_PROCEEDING | GCMSK_PROC_SEND |
GCMSK_PROGRESS| GCMSK_SETUP_ACK,GCACT_SUBMSK);
        if (ret != GC_SUCCESS) {
                cout << " disable Evt not. failed. " << endl;
                return false;
        }
*/
/*      ret = gc_SetEvtMsk(cHdl, GCMSK_BLOCKED|GCMSK_UNBLOCKED,GCACT_ADDMSK);
        if (ret != GC_SUCCESS) {
                return false;
        }
*/
        return true;
}
long
GlobalCallTChannel::dt_handler(unsigned long evhandle)
{
        long chdesc = sr_getevtdev(evhandle);
        long chEventType = sr_getevttype(evhandle);
        GlobalCallTChannel *pChannel = NULL;

        if (chEventType & DT_GC) {
                pChannel = GlobalCallTSP::GetInstance()->
                        FindChannelForDescriptor(chdesc);
                if (pChannel) {
                        int ret = pChannel->LoadMetaEventData();
                        GCTSPLog::diag("Channel \"%ld\" found for device %s.", pChannel->GetHdl(),
ATDV_NAMEP((int)chdesc));
                        if (ret != GC_SUCCESS) {
                                GCTSPLog::error("LoadMetaEventData failed." );
                        }
```

```
                }
        }
        else if (chEventType & DT_DTI) {
                GCTSPLog::warning("Unexpected DTI event 0x%X on device %s (%ld)",
                        chEventType, ATDV_NAMEP((int)chdesc), chdesc);
        }
        // ignore Call Status Transition (CST) and other media (TDX) events
        if (!pChannel &&
                ( (chEventType >= TDX_PLAY) &&
                  (chEventType <= TDX_NOSTOP)
                )
          ) {
                return 1;
        }
        if (!pChannel || !pChannel->HandleEvent(chEventType)) {
                GCTSPLog::warning("Channel \"%ld\" not found for device %s.", chdesc,
ATDV_NAMEP((int)chdesc));
                GCTSPLog::warning("Unhandled GlobalCall event 0x%X on device %s (%ld)",
                        chEventType, ATDV_NAMEP((int)chdesc), chdesc);
        }
        return 1;
}
// virtual
bool
GlobalCallTChannel::HandleEvent(unsigned long event)
{
   SessionNode* session = cSession;
   Monitor m;
   Synchronized sync( session != NULL ? session : &m);
        debug("Got GlobalCall event %ld", event);
        if (GCEV_TASKFAIL == event) {
                gm::string msgHeader = " Got GCEV_TASKFAIL";
                CheckErrorUsingResultValue(msgHeader);
     int ret;
     if ((ret = gc_ResetLineDev(cHdl, EV_ASYNC)) != GC_SUCCESS) {  // disconnect any active calls on the line
device
       CheckError(ret, "gc_ResetLineDev");
     }
        }
        if (GCEV_SETCHANSTATE == event) {
                SetChannelState();
        }
        else if (GCEV_DISCONNECTED == event) {
                bool    moreErrorDetail = true;
                int             gc_result;
                int             cclibid;
                long    ccresult;
                char    *resMsg;
     if (gc_GetMetaEvent(&cMetaevent) != GC_SUCCESS) {
                        moreErrorDetail = false;
                        error("%s gc_GetMetaEvent failed.", GetCallState());
                }
                else {
                        if (gc_ResultValue(&cMetaevent, &gc_result, &cclibid, &ccresult) != GC_SUCCESS) {
                                moreErrorDetail = false;
                                error("%s gc_ResultValue failed.!", GetCallState());
```

```
                }
                else {
                        if (gc_ResultMsg(cclibid, ccresult, &resMsg) != GC_SUCCESS) {
                                // moreErrorDetail is not set here - this is intentional
                                // ccresult is already obtained
                                error("%s gc_ResultMsg failed.", GetCallState());
                        }
                        else {
                                info("%s Disconnect Cause - %s.", GetCallState(), resMsg);
                        }
                }
        }

        if (cCurrentRequestToken.IsValid()) {
                // since current request is valid, this means we're in the process of making an outgoing call
                cMyCallState = eNullState;
                // handle the different cases of call progress problems
                CallProgressNotification::CallProgressT notification =
                                CallProgressNotification::CPN_NetworkError;
                if (moreErrorDetail) {
                        switch(ccresult) {
                                case ERR_ISDN_CAUSE + 0x10:  // Normal Clearing
                                case ERR_ISDN_CAUSE + 0x13:  // No Answer
                                case ERR_ISDN_CAUSE + 0x1F:  // Normal, unspecified
                                        notification = CallProgressNotification::CPN_NoAnswer;
                                        break;
                                case ERR_ISDN_CAUSE + 0x11: // User Busy
                                case ERR_ISDN_CAUSE + 0x12: // No User Responding
                                        notification = CallProgressNotification::CPN_UserBusy;
                                        break;
                                case ERR_ISDN_CAUSE + 0x22: // No Circuit Available
                                        notification = CallProgressNotification::CPN_NetworkBusy;
                                        break;
                                case ERR_ISDN_CAUSE + 0x16: // Number Changed
                                case ERR_ISDN_CAUSE + 0x1C: // Invalid Number Format
                                        notification = CallProgressNotification::CPN_InvalidNumber;
                                        break;
                                case ERR_ISDN_CAUSE + 0x15: // Call Rejected
                                        notification = CallProgressNotification::CPN_CallRejected;
                                        break;
                                default:
                                        break;
                         }
                }
                NotifyCallProgress(notification);
DropCall();
                //ReleaseCall();
                //NotifyCallDropped();
                //NotifySettled();
                info("%s Network is terminating call..., Client has been notified", GetCallState());
         }
         else {
                // this happens when incoming call was cancelled or the far end hangs up
                // OR we may get this event as an unsolicited event after gc_DropCall()
                // is issued.
                // If the last one happens, we need to ignore it to avoid calling gc_DropCall() again
```

```
                        if (cMyCallState == eIdleState || cMyCallState == eNullState) {
                                // unsolicited GCEV_DISCONNECTED event, ignore it
                                diag("%s GCEV_DISCONNECTED received - ignoring this unsolicited event after
gc_DropCall is issued.", GetCallState());
                        }
                        else {
                                info("%s GCEV_DISCONNECTED received - Far End hangs up or cancels the
call.", GetCallState());
                                return handleDisconnectedEvents(GCEV_DISCONNECTED);
                        }
                }
        }
        else if (GCEV_BLOCKED == event) {
                cStatus = S_OutOfService;
                info("%s GCEV_BLOCKED received -- Set the channel status to \"Out of Service\".",
GetCallState());
                // Handle BLOCKED lines by going to NULL state and waiting for UNBLOCKED
                switch(cMyCallState) {
                        case eOfferedState:
                                info("%s GCEV_BLOCKED event received.", GetCallState());
                        case eAcceptedState:
                        case eConnectedState:
                        case eDialingState:
                        case eAlertingState:
                                return handleDisconnectedEvents(GCEV_DISCONNECTED);
                        case eNullState:
                        case eIdleState:
                        case eDisconnectedState:
                                return handleNullEvents(event);
                        default:
                                error("%s GCEV_BLOCKED event received - state not handled.", GetCallState());
                                break;
                }
        }
        else { // after gc_Open(), wait for the GCEV_UNBLOCKED event first before any call
                switch(cMyCallState) {
                        case eNullState:
                                return handleNullEvents(event);
                        case eOfferedState:
                                return handleOfferedEvents(event);
                        case eAcceptedState:
                                return handleAcceptedEvents(event);
                        case eConnectedState:
                                return handleConnectedEvents(event);
                        case eDialingState:
                                return handleDialingEvents(event);
                        case eAlertingState:
                                return handleAlertingEvents(event);
                        case eDisconnectedState:
                                return handleDisconnectedEvents(event);
                        case eIdleState:
                                return handleIdleEvents(event);
                        default:
                                error("%s GCEV_UNBLOCKED received -- state not handled.", GetCallState());
                                break;
                }
```

```
        }

        return true;
}
// virtual
bool
GlobalCallTChannel::handleNullEvents(int event)
{
        cMyCallState = eNullState;
        cIncomingCallValid = true;
        switch (event)
        {
                case GCEV_UNBLOCKED:
                {
                        cTimeChannelReleased = ::GetTickCount();
                        diag("%s GCEV_UNBLOCKED received.", GetCallState());
                        cReady = true;
                        info("%s Telephony Channel is ready.", GetCallState());
                        SetChannelState();
                        // go to the GCEV_RESETLINEDEV case
                }
                case GCEV_RESETLINEDEV:
                {
                        cMyCallState = eNullState;// Still in Null state until Offered event
                        if (GCEV_RESETLINEDEV == event) {
                                diag("%s GCEV_RESETLINEDEV received.", GetCallState());
                                cReady = true;          // just for testing - to be removed after test
                                SetChannelState();  // just for testing - to be removed after test
                        }
        info("%s Notifying that channel is settled", GetCallState());
        NotifySettled();
        info("%s channel is settled, wait for the next incoming call", GetCallState());
                        return WaitForIncomingCall();
                }
                case GCEV_OFFERED:
                        // Make sure we have CRN info
        if (gc_GetCRN(&cCrnData, &cMetaevent) != GC_SUCCESS)
                                return false;
                        return handleOfferedEvents(GCEV_OFFERED);
                case GCEV_BLOCKED:
                {
                        diag("%s handleNullEvents - GCEV_BLOCKED received.", GetCallState());
                        break;
                }
                default:
                {
                        warning("%s handleNullEvents - got unexpected event 0x%X", GetCallState(), event);
                        int lineDevStatus;
                        GetBChannelStatus(&lineDevStatus);
                        break;
                }
         }
         return true;
}
//virtual
bool
```

```
GlobalCallTChannel::handleOfferedEvents(int event)
{
   debug("Setting state to eOfferedState");
   cMyCallState = eOfferedState;
   debug("Set state to eOfferedState");
        switch (event) {
                case GCEV_OFFERED:
                {
                        diag("%s GCEV_OFFERED received.", GetCallState());
            cANIBuf[0] = '\0';
                        if (gc_GetANI(cCrnData, cANIBuf) != GC_SUCCESS) {
                                error("%s Couldn't get ANI", GetCallState());
                        } // continue for now
            cDNISBuf[0] = '\0';
                        if (gc_GetDNIS(cCrnData, cDNISBuf) != GC_SUCCESS) {
                                error("%s Couldn't get DNIS", GetCallState());
                        } // continue for now
                        unsigned long ret;
        USRINFO_ELEM usrinfo;
                        bool uuifound = false;
                        int i = 0, pos = 0;
        if (ret = gc_GetCallInfo(cCrnData, U_IES, (char *)&usrinfo))
        {
          CheckError(ret, "gc_GetCallInfo");
                                error("%s Couldn't get Call Info", GetCallState());
        }
        else
        {
                                if (usrinfo.length > 1)
                                {
                                        while (!uuifound && i<usrinfo.length)
                                        {
                                                if (usrinfo.usrinformation[i] == USER_USER_IE)
                                                {
                                                        uuifound = true;
                                                        pos = i;
                                                }
                                                else
                                                {
                                                        if (usrinfo.usrinformation[i+1] < 0)
                                                                break;
                                                        i += usrinfo.usrinformation[i+1] + 2;
                                                }
                                        }
                                }
        }

#if 0
        GC_IE_BLK gcIeBlk;
        GCLIB_IE_BLK gclibIeBlk;
        IE_BLK ieBlk;
        gcIeBlk.gclib = &gclibIeBlk;
        gcIeBlk.cclib = &ieBlk;
        unsigned long ret;
        if ((ret = gc_GetInfoElem(cHdl, &gcIeBlk)) != GC_SUCCESS)
        {
```

```
        CheckError(ret, "gc_GetInfoElem");
                        error("%s Couldn't get UUI by using gc_GetInfoElem", GetCallState());
    }
    else
    {
    }

#endif
                info("%s Call offered ANI=\"%s\", DNIS=\"%s\"", GetCallState(), cANIBuf, cDNISBuf);
                if (!cCurrentCallToken.IsValid()) {
                        if (acceptInboundCall())  { // grab it immediately
                                InitializeCurrentCallToken();
                                cCurrentCallToken.SetSpan(cBoardNum);
                                cCurrentCallToken.SetChannel(cPortNum);
                                AssignSequenceNumber();
                                cCurrentCallToken.SetTSPPrivate(cHdl);
                                Address DNIS(cDNISBuf);
                                Address ANI(cANIBuf);
                                info("%s Notify Client of Incoming Call", GetCallState());
                                NotifyIncomingCall(DNIS, ANI, uuifound ?
&usrinfo.usrinformation[pos+3] : 0, uuifound ? usrinfo.usrinformation[pos+1]-1 : 0);
                        }
                        else {
                                cIncomingCallValid = false;
                                info("%s Incoming Call becomes invalid.", GetCallState());
                                // this means the incoming call cannot be grabbed
                                // if the disconnect event comes when this cIncomingCallValid is false,
                                // we should not let app know since it does not care about it
                        }
                }
                else {
                        info("%s Current Call Token valid - cannot notify incoming call.", GetCallState());
                }
                break;
        }
        case GCEV_ACCEPT:
                diag("%s GCEV_ACCEPT received.", GetCallState());
                return handleAcceptedEvents(GCEV_ACCEPT);
        case GCEV_ANSWERED:
                diag("%s GCEV_ANSWERED received.", GetCallState());
                return handleConnectedEvents(GCEV_ANSWERED);
        case GCEV_RESETLINEDEV:
                diag("%s GCEV_RESETLINEDEV received", GetCallState());
                break;
        case GCEV_UNBLOCKED:
                diag("%s GCEV_UNBLOCKED received.", GetCallState());
                SetChannelState();
                break;
        case GCEV_BLOCKED:
                diag("%s in handleOfferedEvents - GCEV_BLOCKED received.", GetCallState());
                break;
        default:
                warning("%s in handleOfferedEvents - got unexpected event 0x%X", GetCallState(), event);
                break;
    }
    return true;
```

```
}
//*************************************************************************/
//virtual
bool
GlobalCallTChannel::handleAcceptedEvents(int event)
{
        cMyCallState = eAcceptedState;
        switch (event) {
                case GCEV_ACCEPT:
                        info("%s Inbound Call Accept Acknowledgment.", GetCallState());
                        // this is an acknowledgement it's accepted - don't do anything
                        break;
                case GCEV_ANSWERED:
                        return handleConnectedEvents(GCEV_ANSWERED);
                default:
                        warning("%s handleAcceptedEvents - got unexpected event 0x%X", GetCallState(), event);
                        break;
        }
        return true;
}
//*************************************************************************/
//virtual
bool
GlobalCallTChannel::handleDialingEvents(int event)
{
        cMyCallState = eDialingState;
        switch (event) {
                case GCEV_CONNECTED:
                        diag("%s GCEV_CONNECTED received.", GetCallState());
                        return handleConnectedEvents(GCEV_CONNECTED);
                case GCEV_ALERTING:
                        diag("%s GCEV_ALERTING received.", GetCallState());
                        return handleAlertingEvents(GCEV_ALERTING);
                case GCEV_PROCEEDING:
                        diag("%s GCEV_PROCEEDING received.", GetCallState());
                        break;
                case GCEV_PROGRESSING:
                        diag("%s GCEV_PROGRESSING received.", GetCallState());
                        break;
                case GCEV_TASKFAIL:
                        error("%s GCEV_TASKFAIL received.", GetCallState());
                        // notify app
                        if (cCurrentRequestToken.IsValid()) {
                                cMyCallState = eNullState;
                                error("%s, Can't dial - GlobalCallTChannel problem with the local end.",
GetCallState());  // not network error
                                NotifyCallProgress(CallProgressNotification::CPN_Error);
                        }
                        break;
                default:
                        warning("%s handleDialingEvents - got unexpected event 0x%X", GetCallState(), event);
                        break;
        }
        return true;
}
//*************************************************************************/
```

```
//virtual
bool
GlobalCallTChannel::handleAlertingEvents(int event)
{
        cMyCallState = eAlertingState;
        switch (event) {
                case GCEV_ALERTING:
            InitializeCurrentCallToken();
            cCurrentCallToken.SetSpan(cBoardNum);
            cCurrentCallToken.SetChannel(cPortNum);
            AssignSequenceNumber();
            cCurrentCallToken.SetTSPPrivate(cHdl);
       NotifyCallProgress(CallProgressNotification::CPN_Alerting);
                        break;
                case GCEV_CONNECTED:
                        return handleConnectedEvents( GCEV_CONNECTED );
                default:
                        warning("%s handleAlertingEvents - got unexpected event 0x%X", GetCallState(), event);
                        break;
         }
        return true;
}
//*************************************************************************/
//virtual
bool
GlobalCallTChannel::handleConnectedEvents(int event)
{
        cMyCallState = eConnectedState;
        switch (event) {
     case GCEV_ANSWERED:
                        //::SetEvent(cInCallConnectedEvt);
                        diag("%s GCEV_ANSWERED received - handleConnectedEvents.", GetCallState());
        {
          Synchronized sync(cSession);
          sync.notifyAll();
        }
                        // notify app that call is connected
                        //printMessage("GlobalCallTChannel",
                        //       "GCEV_ANSWERED received - notifying Inbound Call Connected." );
                        // set the "inbound call connected event"
                        //NotifyCallConnected();
                        break;
                case GCEV_CONNECTED:
        if (!cCurrentCallToken.IsValid())
        {
                           InitializeCurrentCallToken();
                           cCurrentCallToken.SetSpan(cBoardNum);
                           cCurrentCallToken.SetChannel(cPortNum);
                           AssignSequenceNumber();
                           cCurrentCallToken.SetTSPPrivate(cHdl);
        }
                        info("%s GCEV_CONNECTED received - About to notify client of Call Connected.",
GetCallState());
                        NotifyCallConnected();
                        break;
                default:
```

```
                        warning("%s handleConnectedEvents - got unexpected event 0x%X", GetCallState(),
event);
                        break;
        }
        return true;
}
//*****************************************************************************
//virtual
bool
GlobalCallTChannel::handleDisconnectedEvents(int event)
{
        if (cIncomingCallValid)
                cMyCallState = eDisconnectedState;
        else {
                cMyCallState = eIdleState;
                cIncomingCallValid = true;
                warning("%s Cannot disconnect -- call has not been connected.", GetCallState());
                return true; // this must be a disconnect that comes when call has not been connected
        }
        switch (event) {
                case GCEV_DISCONNECTED: {
                        // Line has been dropped, clean up.
                        diag("%s In handleDisconnectedEvents, calling NotifyAboutToDropCall.", GetCallState());

                        NotifyAboutToDropCall();
                        diag("%s In handleDisconnectedEvents, called NotifyAboutToDropCall, calling DropCall.",
GetCallState());
                        unsigned long ret = DropCall();
                        info("%s In handleDisconnectedEvents, called DropCall to disconnect.", GetCallState());
                        if (ret == TSPF_CALLALREADYDISCONNECTED) {
                                info("%s In handleDisconnectedEvents, Call already disconnected.",
GetCallState());
                                return false;
                        }
                        else if (ret != TSPF_OK) {
                                // since we'll get neither
                                // GCEV_TASKFAIL nor GCEV_DROPCALL
                                // handle as if we were getting GCEV_TASKFAIL
                                error("%s In handleDisconnectedEvents, Failed to drop call.", GetCallState());
                                return handleIdleEvents(GCEV_TASKFAIL);
                        }
                        break;
                }
                case GCEV_DROPCALL:
                        diag("%s GCEV_DROPCALL received - handleDisconnectedEvents.", GetCallState());
                        return handleIdleEvents(GCEV_DROPCALL);
                default:
                        warning("%s handleDisconnectedEvents - got unexpected event 0x%X", GetCallState(),
event);
                        break;
        }
        return true;
}
//*****************************************************************************/
//virtual
bool
```

```
GlobalCallTChannel::handleIdleEvents(int event)
{
        cMyCallState = eIdleState;
        cIncomingCallValid = true;
        switch (event) {
                case GCEV_TASKFAIL:
                        // must be because gc_DropCall does not complete
                        // handle it as if GCEV_DROPCALL is received but do not call ReleaseCall
                        error("%s handleIdleEvents -- GCEV_TASKFAIL received.", GetCallState());
                case GCEV_DROPCALL:
                        if (GCEV_DROPCALL == event) {
                                diag("%s handleIdleEvents -- GCEV_DROPCALL received.", GetCallState());
                                if (ReleaseCall() != 0) {
                                        error("%s handleIdleEvents -- Release Call failed.", GetCallState());
                                }
                                else {
                                        cTimeChannelReleased = ::GetTickCount();
                                }
                        }
                        // set up for inbound calls
                        cMyCallState = eNullState;
                        // cNotifyCallDropped is set to true only in conjuction to
                        // a call to gc_DropCall()
                        if (cNotifyCallDropped) {
                                diag("%s handleIdleEvents -- calling NotifyCallDropped.", GetCallState());
                                NotifyCallDropped();
                                diag("%s handleIdleEvents -- called NotifyCallDropped.", GetCallState());
                                cNotifyCallDropped = false;
                                SessionNode* s = cSession;
                                if (s != NULL) {
                         Synchronized sync(s);
                               sync.notifyAll();
                                }
                                //::ResetEvent(cInCallConnectedEvt);
                        }

                        NotifySettled();
                        // no need to call WaitForIncomingCall here - only need to call gc_WaitCall once when
using the asynchronous mode
        break;
                case GCEV_ACCEPT:
                case GCEV_ANSWERED:
                        // this means application drops the call
                        // when an incoming call is in progress (transitioned to idle state)
                        info("%s handleIdleEvents - Got either GCEV_ANSWERED or GCEV_ACCEPT event
(0x%X) when call is already dropped by client - ignoring it.", GetCallState(), event);
                        break;
                case GCEV_PROCEEDING:
                case GCEV_PROGRESSING:
                case GCEV_ALERTING:
                        info("%s handleIdleEvents - Got either GCEV_PROCEEDING, GCEV_ALERTING or
GCEV_PROGRESSING event (0x%X) when call is already cancelled - ignoring it.", GetCallState(), event);
                        break;
                default:
                        warning("%s handleIdleEvents - got unexpected event 0x%X", GetCallState(), event);
                        break;
```

```
        }
        return true;
}
int
GlobalCallTChannel::LoadMetaEventData(void)
{
  return gc_GetMetaEvent(&cMetaevent);
}
bool
GlobalCallTChannel::WaitForIncomingCall(void)
{
        if (!cWaitingForCall) {
                /* Wait for an incoming call */
                int ret = gc_WaitCall( cHdl,                    /* Line Device */
                                                                  NULL,                 /* crnp      */
                                                                  NULL,                 /* waitcallp  */
                                                                  0,                    /* timeout    */
                                                                  EV_ASYNC    /* mode       */
                                                                 );

                if (ret != GC_SUCCESS) {
                        return false;
                }
                info("%s Wait for incoming call...", GetCallState());
                cWaitingForCall = true;
        }
        return true;
}
// virtual
unsigned long
GlobalCallTChannel::PlaceCall(const gm::string& calledNumber,
                                                                  const gm::string& callingNumber, unsigned char *uui,
int uuiLength)
{
/* test code
        static int i=1;
        if ((i++ % 2) == 0) {
                SetStatusAndNotifyTSP(TelephonyChannel::S_OutOfService);
                printMessage("GlobalCallTChannel",
                        "Error: gc_MakeCall() -- globalcall error -- setting channel status Out of Service.\n");
                return TSPF_SYSTEMFAILURE;
        }
*/
        if (!cReady || !IsAlive()) {
                error("%s Can't dial - cReady is %d.", GetCallState(), cReady);
                return TSPF_CANTDIAL;
        }
        if (uui && uuiLength == 0)
                warning("uui length is 0");
        if (uui && (uuiLength < 0 || uuiLength > MAXLEN_IEDATA))
        {
                error("UUI length exceeded the maximum, can't dial");
                return TSPF_CANTDIAL;
        }
   GC_MAKECALL_BLK      gcMcBlk;
        MAKECALL_BLK    mcBlk;
```

```
  // Setup the MakeCall Block for ISDN_UIL1_G711ULAW
        // - this is somehow needed for a successful ISDN outbound call;
  memset(&gcMcBlk, 0xff, sizeof(GC_MAKECALL_BLK));
  memset(&mcBlk, 0xff, sizeof(MAKECALL_BLK));
        gcMcBlk.gclib = NULL;
        gcMcBlk.cclib = &mcBlk;
  mcBlk.isdn.BC_xfer_cap = BEAR_CAP_SPEECH;
  mcBlk.isdn.BC_xfer_mode = ISDN_ITM_CIRCUIT;
  mcBlk.isdn.BC_xfer_rate = BEAR_RATE_64KBPS;
  mcBlk.isdn.usrinfo_layer1_protocol = ISDN_UIL1_G711ULAW;
  mcBlk.isdn.destination_number_type = 0xff;
  mcBlk.isdn.destination_number_plan = ISDN_NUMB_PLAN;
  mcBlk.isdn.origination_number_type = NAT_NUMBER;
  mcBlk.isdn.origination_number_plan = ISDN_NUMB_PLAN;
  strcpy(&(mcBlk.isdn.origination_phone_number[0]), callingNumber == 0 ? "" : (char *)(const char
*)callingNumber);
  mcBlk.isdn.facility_feature_service = 0xff;
  mcBlk.isdn.facility_coding_value = 0xff;
#if 0
  // set user-to-user info
        GC_IE_BLK     gcIeBlk;
        GCLIB_IE_BLK   gclibIeBlk;
        IE_BLK        ieBlk;
        gcIeBlk.gclib = NULL; //&gclibIeBlk;
        gcIeBlk.cclib = &ieBlk;
        if (uui && uuiLength > 0)
        {
                ieBlk.length = uuiLength;
/*              ieBlk.data[0] = 0x7E;
                ieBlk.data[1] = 0x8;
                ieBlk.data[2] = 0x1;
                ieBlk.data[3] = 0x2;
                ieBlk.data[4] = 0x3;
                ieBlk.data[5] = 0x4;
*/
                memcpy(ieBlk.data, (char *)uui, uuiLength);
                unsigned long ret = 0L;
                if (ret = gc_SetInfoElem(cHdl, &gcIeBlk) < 0)
                {
                        // error
                        CheckError(ret, "gc_SetInfoElem");
                }
                else
                {
                        debug("UUI set up");
                }
        }
        mcBlk.isdn.usrinfo_bufp = 0;
#endif
//#if 0
        USRINFO_ELEM usrinfo;
        if (uui && uuiLength > 0)
        {
                usrinfo.length = uuiLength+1;
                usrinfo.protocol_discriminator = 'i';
```

```
                memcpy(usrinfo.usrinformation, (char *)uui, uuiLength);

                mcBlk.isdn.usrinfo_bufp = &usrinfo;
        }
        else
                mcBlk.isdn.usrinfo_bufp = 0;
//#endif
        if (gc_SetCallingNum(cHdl, callingNumber == 0 ? "" : (char *)(const char *)callingNumber)
                != GC_SUCCESS) {
                error("%s Can't place call - Can't set calling number.", GetCallState());
                return TSPF_CANTDIAL;
        }
        int             err = GC_SUCCESS;
        int             libId;
        long    libErr;
        char    *pErrMsg;
        unsigned long tNow = ::GetTickCount();
        unsigned long tDiff = tNow - cTimeChannelReleased;
        debug("%s About to make a call, time channel back in service = %ld, tNow = %ld, tDiff = %ld",
GetCallState(), cTimeChannelReleased, tNow, tDiff);
#if 0
        if ((tDiff != tNow) && (tDiff >= (unsigned long)0L) && (tDiff < WAIT_TIME_BEFORE_DIALING)) {
                debug("%s Sleep %d msecs before next call.", GetCallState(), WAIT_TIME_BEFORE_DIALING -
tDiff);
                Sleep(WAIT_TIME_BEFORE_DIALING - tDiff);
        }
#endif
        cTimeChannelReleased = 0L;
        if ((err = gc_MakeCall( cHdl,
                                        &cCrnData,
                                                (char *)(const char *)calledNumber,
                                                &gcMcBlk,
                                                30,
                                                EV_ASYNC
                                ))
                                        != GC_SUCCESS) {
                error("%s Can't dial - gc_MakeCall failed.", GetCallState());
                if (gc_ErrorValue(&err, &libId, &libErr) != GC_SUCCESS) {
                        error("%s Could not get error value -- gc_ErrorValue failed.", GetCallState());
                }
                else {
                        if (gc_ResultMsg( libId, (long)err, &pErrMsg ) != GC_SUCCESS) {
                                error("%s Could not get error message -- gc_ResultMsg failed.", GetCallState());
                        }
                        else {
                                error("%s gc_MakeCall() error: 0x%lx, %s.", GetCallState(), (long)err, pErrMsg);
                        }
                }
                // set channel out of service
                if (err == EGC_LINERELATED)
                {
                        SetStatusAndNotifyTSP(TelephonyChannel::S_OutOfService);
                        error("%s gc_MakeCall() error -- set channel status to Out of Service.", GetCallState());
                }
                return TSPF_SYSTEMFAILURE;
        }
```

```
        cMyCallState = eDialingState;
        info("%s Call has been made from \"%s\" to \"%s\".", GetCallState(),
                callingNumber.c_str(), calledNumber.c_str());
        return TSPF_OK;

}
// virtual
unsigned long
GlobalCallTChannel::CancelRequest(void)
{
        info("%s Call Cancel has been requested.", GetCallState());
        if (cCurrentRequestToken.IsValid() && cMyCallState == eNullState) { // failure in dialing
                info("%s In CancelRequest: cancelling call request for a call attempt that failed.", GetCallState());
                NotifyCallDropped();
                NotifySettled();
                return TSPF_OK;
        }
        if (cMyCallState == eDialingState ||
                cMyCallState == eAlertingState ||
                cMyCallState == eDisconnectedState) {
                unsigned long ret = DropCall();
                if (ret != TSPF_OK) {
                        if (ret == TSPF_CALLALREADYDISCONNECTED) {
                                warning("%s In CancelRequest: Can't cancel - DropCall() returns
TSPF_CALLALREADYDISCONNECTED.", GetCallState());
                                return ret;
                        }
                        else {
                                warning("%s In CancelRequest: Can't cancel - DropCall() returns some failure
status.", GetCallState());
                                return TSPF_CANTCANCEL;
                        }
                }
                else {
        Synchronized sync(cSession);
        cSession->SetGoingAway(true);
                        info("%s In CancelRequest: DropCall() succeeded.", GetCallState());
                        return TSPF_OK;
                }
         }
         else {
                warning("%s Can't cancel because channel state is not eDialingState, eAlertingState or
eDisconnectedState.", GetCallState());
                return TSPF_CANTCANCEL;
        }
}
// virtual
unsigned long
GlobalCallTChannel::DropCall(void)
{
        if (cMyCallState == eIdleState) {
                info("%s In DropCall: cannot drop the call because the channel state is idle - call was already
dropped.", GetCallState());
                return TSPF_CALLALREADYDISCONNECTED;
        }
        else {
```

```
                return disconnectCall(GC_NORMAL_CLEARING);
        }
}
unsigned long
GlobalCallTChannel::ReleaseCall()
{
        unsigned long ret = 0; // success
        diag("%s Calling gc_ReleaseCall() to release call.", GetCallState());
        ret = gc_ReleaseCall(cCrnData);
        diag("%s Called gc_ReleaseCall().", GetCallState());
  // we may return error code in the future
        if (ret == GC_SUCCESS) {
                info("%s gc_ReleaseCall() succeeded - Call released.", GetCallState());
                return 0;
        }
        else {
                error("%s gc_ReleaseCall() failed.", GetCallState());
                return -1;
        }
}
// virtual
unsigned long
GlobalCallTChannel::ListenTo(unsigned short timeslot)
{
        if (timeslot != (unsigned short)-1)
        {
                SC_TSINFO tsinfo;
                long timeslots = timeslot;
                info("%s Making this Telephony Channel listen to timeslot %d",
                        GetCallState(), timeslot);

                tsinfo.sc_numts = 1;
                tsinfo.sc_tsarrayp = &timeslots;
                if (dt_listen(cHdl, &tsinfo) == -1)
                {
                        error("%s dt_listen failed!", GetCallState());
                        return TSPF_CANTLISTEN;
                }
        }
        else
        {
                info("%s Unlistening this Telephony Channel.", GetCallState());
                if (dt_unlisten(cHdl) == -1)
                {
                        error("%s dt_unlisten failed!", GetCallState());
                        return TSPF_CANTLISTEN;
                }
        }
        return TSPF_OK;
}
// virtual
unsigned long
GlobalCallTChannel::PlaceCallOnHold(void)
{
        //
        // DFB_TBD
```

```
        //
        return TSPF_OK;
}
// virtual
unsigned long
GlobalCallTChannel::AcceptCall(void)
{
  Synchronized sync(cSession);
        //::SetEvent(cWaitingForTerminationAckEvt);
        if (gc_AnswerCall(cCrnData, 0, EV_ASYNC) != GC_SUCCESS) {
        //      ::ResetEvent(cWaitingForTerminationAckEvt);
                return TSPF_CANTANSWER;
        }
  while (cMyCallState == eOfferedState || cMyCallState == eAcceptedState) {
    sync.wait();
  }
  return cMyCallState == eConnectedState ? TSPF_OK : TSPF_CANTANSWER;
        //DWORD ret = ::WaitForSingleObject(cInCallConnectedEvt, 2000);
        //::ResetEvent(cWaitingForTerminationAckEvt);
        //if (ret != WAIT_OBJECT_0)
        //      return TSPF_CANTGOOFFHOOK;
}
bool
GlobalCallTChannel::acceptInboundCall()
{
        return (gc_AcceptCall(cCrnData, 0, EV_ASYNC) == GC_SUCCESS ?
                                        true :
                                        false);
}
// virtual
unsigned long
GlobalCallTChannel::RejectCall(TelephonyChannel::RejectCause cause)
{
        info("%s Rejecting call... ", GetCallState());
        //return disconnectCall(GC_UNASSIGNED_NUMBER);
        if (cMyCallState != eAcceptedState && cMyCallState != eOfferedState)
                return TSPF_CANTGOONHOOK;
  Synchronized sync(cSession);
  cSession->SetGoingAway(true);
        return disconnectCall(GC_USER_BUSY);
}
// virtual
unsigned long
GlobalCallTChannel::disconnectCall(int cause)
{
  int ret;
        diag("%s In DropCall: calling gc_DropCall().", GetCallState());
        // notify client of "call dropped" later when
        // CRN is released (after the call to gc_Release())
        cNotifyCallDropped = true;
  if ((ret = gc_DropCall(cCrnData, cause, EV_ASYNC))
                == GC_SUCCESS) {
                cMyCallState = eIdleState;
                diag("%s in disconnectCall() - called gc_DropCall() - succeeded.", GetCallState());
                return TSPF_OK;
        }
```

```
        else {
                cNotifyCallDropped = false; // don't notify client
                error("%s in disconnectCall() - called gc_DropCall() - failed.", GetCallState());
                CheckError(ret, "gc_DropCall");
                return TSPF_CANTGOONHOOK;
        }
}
void
GlobalCallTChannel::AssignSequenceNumber(void)
{
        Synchronize sync(cCriticalSection);
        cCurrentCallToken.SetSequenceNumber(cSequenceNumber++);
}
bool
GlobalCallTChannel::IsAlive(void)
{
        int     chanStatus;
        return (GetDChannelStatus(&chanStatus) &&
                        chanStatus == DATA_LINK_UP &&
                        GetBChannelStatus(&chanStatus) &&
                        chanStatus == GCLS_INSERVICE);
}
bool
GlobalCallTChannel::GetBChannelStatus(int *status)
{
        if (gc_GetLinedevState(cHdl, GCGLS_BCHANNEL, status) != GC_SUCCESS) {
                error("%s Couldn't get B Channel status.", GetCallState());
                return false;
        }
        else {
                int i = *status;
                info("%s B Channel Status is %d.", GetCallState(), i);
                return true;
        }
}
bool
GlobalCallTChannel::GetDChannelStatus(int *status)
{
        if (gc_GetLinedevState(cHdl, GCGLS_DCHANNEL, status) != GC_SUCCESS) {
                error("%s Couldn't get D Channel status.", GetCallState());
                return false;
        }
        else {
                int i = *status;
                info("%s D Channel Status is %d.", GetCallState(), i);
                return true;
        }
}
void
GlobalCallTChannel::SetChannelState(void)
{
        // Set channel status according to the B channel status.
   if (GetCapabilities() & TelephonyChannel::C_Unused)
   {
                cStatus = S_OutOfService;
                info("Unused Channel -- Set status to \"Out of Service\".");
```

```
    return;
  }
        int BLineDevStatus;
        int DLineDevStatus;
        bool bBStatusGood = GetBChannelStatus(&BLineDevStatus);
        bool bDStatusGood = GetDChannelStatus(&DLineDevStatus);
        if (bDStatusGood && DLineDevStatus == DATA_LINK_UP &&
                bBStatusGood && BLineDevStatus == GCLS_INSERVICE) {
                cStatus = S_InService;
#if 0
                NotifyChannelAvailable();
#endif
                info("%s B channel = GCLS_INSERVICE -- Set the channel status to \"In Service\".",
GetCallState());
        }
        else {
                cStatus = S_OutOfService;
                info("%s B channel != GCLS_INSERVICE -- Set the channel status to \"Out of Service\".",
GetCallState());
        }
}
void
GlobalCallTChannel::CheckErrorUsingResultValue(const gm::string& messageHeader)
{
        // print what task failed
        int gc_result;
        int cclibid;
        long ccresult;
        char *resMsg;
  if (gc_GetMetaEvent(&cMetaevent) != GC_SUCCESS) {
                error("%s %s -- gc_GetMetaEvent failed.",
                        GetCallState(), (char *)(const char *)messageHeader);
        }
        else {
                if (gc_ResultValue(&cMetaevent, &gc_result, &cclibid, &ccresult)
                        != GC_SUCCESS) {
                        error("%s %s -- gc_ResultValue failed.",
                                GetCallState(), (char *)(const char *)messageHeader);
                }
                else {
                        error("%s %s -- err code (gc): %d, err code (cc): 0x%lx.",
                                GetCallState(), (char *)(const char *)messageHeader, gc_result, ccresult);
                        if (gc_ResultMsg(cclibid, ccresult, &resMsg) != GC_SUCCESS) {
                                error("%s %s -- gc_ResultMsg failed.",
                                        GetCallState(), (char *)(const char *)messageHeader);
                        }
                        else {
                                error("%s %s -- error message: %s",
                                        GetCallState(), (char *)(const char *)messageHeader, resMsg);
                        }
                }
        }
}
const char const *
GlobalCallTChannel::GetCallState(void)
{
```

```
        switch(cMyCallState) {
                case eNullState:
                        return cStates[0];
                case eDisconnectedState:
                        return cStates[1];
                case eIdleState:
                        return cStates[2];
                case eOfferedState:
                        return cStates[3];
                case eAcceptedState:
                        return cStates[4];
                case eDialingState:
                        return cStates[5];
                case eConnectedState:
                        return cStates[6];
                case eAlertingState:
                        return cStates[7];
                default:
                        warning("In getMyCallStateString -- unknown call channel state %d", (int)cMyCallState);
                        return cStates[8];
        }
}
void
GlobalCallTChannel::GetMyCallStateString(gm::string& myCallStateStr)
{
        switch(cMyCallState) {
                case eNullState:
                        myCallStateStr = "Null State";
                        break;
                case eDisconnectedState:
                        myCallStateStr = "Disconnected State";
                        break;
                case eIdleState:
                        myCallStateStr = "Idle State";
                        break;
                case eOfferedState:
                        myCallStateStr = "Offered State";
                        break;
                case eAcceptedState:
                        myCallStateStr = "Accepted State";
                        break;
                case eDialingState:
                        myCallStateStr = "Dialing State";
                        break;
                case eConnectedState:
                        myCallStateStr = "Connected State";
                        break;
                case eAlertingState:
                        myCallStateStr = "Alerting State";
                        break;
                default:
                        warning("In getMyCallStateString -- unknown call channel state %d", (int)cMyCallState);
                        myCallStateStr = "";
                        break;
        }
}
```

```
// virtual
const char *
GlobalCallTChannel::GetIDString(void)
{
        return ATDV_NAMEP(cHdl);
}
void
GlobalCallTChannel::CheckError(int retCode, const char const *funcName)
{
        int             gc_err;
        int             lowerLibID;
        long    lowerLibErr;
        char    *pLibName;
        char    *pErrMsg;
  if (retCode != GC_SUCCESS) {
                if (gc_ErrorValue( &gc_err, &lowerLibID, &lowerLibErr ) != GC_SUCCESS) {
                        error("%s gc_ErrorValue failed in CheckError().", GetCallState());
                }
                else {
                        if (gc_ResultMsg(  LIBID_GC, (long)gc_err, &pErrMsg ) != GC_SUCCESS) {
                                error("%s gc_ResultMsg failed in CheckError().", GetCallState());
                        }
                        else {
                                error("%s %s() failed - GlobalCall Lib err = 0x%lx - %s.",
                                        GetCallState(), funcName, gc_err, pErrMsg);
                        }
                }
                if (lowerLibID != LIBID_GC) {
                        if (gc_ResultMsg(  lowerLibID, (long)lowerLibErr, &pErrMsg ) != GC_SUCCESS) {
                                error("%s gc_ResultMsg failed in CheckError().", GetCallState());
                        }
                        else {
                                if (gc_CCLibIDToName( lowerLibID ,&pLibName ) != GC_SUCCESS) {
                                        error("%s gc_CCLibIDToName failed in CheckError().", GetCallState());
                                }
                                else {
                                        error("%s %s() failed - %s Lib err = 0x%lx - %s.",
                                                GetCallState(), funcName, pLibName, lowerLibErr, pErrMsg);
                                }
                        }
                }
  }
  return;
}
const gm::string&
GlobalCallTChannel::GetChanInfoStr(void)
{
        gm::string stateStr;
        GetMyCallStateString(stateStr);
        char buf[10];
        cChanInfo = "[Dev \"";
        cChanInfo += ATDV_NAMEP((int)cHdl);
        cChanInfo += "\" (";
        _itoa((int)cHdl, buf, 10);
        cChanInfo += buf;
        cChanInfo += ") \"" + stateStr + "\"]";
```

```
        return cChanInfo;
}
void
GlobalCallTChannel::printMessage(const char *group, const char *fmt, ...)
{
        gm::string stateStr;
        GetMyCallStateString(stateStr);
        char buf[10];
  gm::string chanInfo;
        chanInfo = "[Dev \"";
        chanInfo += ATDV_NAMEP((int)cHdl);
        chanInfo += "\" (";
        _itoa((int)cHdl, buf, 10);
        chanInfo += buf;
        chanInfo += ") \"" + stateStr + "\"]";
        char thisaddr[10];
        ::sprintf(thisaddr, "%p", this);
        chanInfo += " ";
        chanInfo += thisaddr;
        chanInfo += " ";
        chanInfo += fmt;
        va_list vl;
        va_start(vl, fmt);
        debug::vlprintf("GlobalCallTChannel", chanInfo, vl);
}
// virtual
void
GlobalCallTChannel::NotifyBoundToSession(SessionNode *pSessionNode)
{
  SCChannel::NotifyBoundToSession(pSessionNode);
        //if (cMyCallState == eNullState)
        //      NotifySettled();
}
bool
GlobalCallTChannel::IsAvailable(void)
{
        SetChannelState();
        return (cMyCallState == eNullState && TelephonyChannel::IsAvailable());
}
bool
GlobalCallTChannel::SetPhysicalChannelStatus(TelephonyChannel::ChannelStatus status)
{
  switch (status)
  {
    case TelephonyChannel::S_InService:
    {
      gc_SetChanState(cHdl, GCLS_INSERVICE, EV_ASYNC);
      break;
    }
    case TelephonyChannel::S_OutOfService:
    {
      gc_SetChanState(cHdl, GCLS_OUT_OF_SERVICE, EV_ASYNC);
      break;
    }
    case TelephonyChannel::S_Maintenance:
    {
```

```
            gc_SetChanState(cHdl, GCLS_MAINTENANCE, EV_ASYNC);
            break;
        }
        default:
        {
            warning("new channel status???");
            break;
        }
    }
    return true; // for now
}
```

```
/*
 * GlobalCallTSP.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.34 $
 * $Date: 2000/12/07 03:28:53 $
 */
static const char* id="$Id: GlobalCallTSP.cpp,v 1.34 2000/12/07 03:28:53 Mitsuru Oshima Exp $";
#include "GlobalCallTSP.h"
#include <windows.h>
#include <stdio.h>
#include <iostream.h>
#include <process.h>
#include <srllib.h>
#include <dtilib.h>
#include <dxxxlib.h>
#include <gclib.h>
#include <gcerr.h>
#include "GlobalCallTChannel.h"
#include "../../Dialogic/MSP/DialogicMChannel.h"
#include "MTSLib/RangeParser.h"
#include "MTSLib/MTSServer.h"
#include "VPPUtils/ConfigMgr.h"
#include "VPPUtils/Synchronize.h"
#include "isdncmd.h"
GlobalCallTSP *GlobalCallTSP::cInstance = 0;
const char const *GlobalCallTSP::protocolName = "P_isdn";
const char const *GlobalCallTSP::cLogType="GLCL_TSP";
typedef unsigned (__stdcall *PBEGINTHREADEX_THREADFUNC) (
                                        LPVOID lpThreadParameter
                                    );
typedef unsigned *PBEGINTHREADEX_THREADID;
GlobalCallTSP::GlobalCallTSP(MTSServer *pServer)
        : TelephonyServiceProvider(pServer, cLogType),
          cThreadH(NULL)
          //, pDeviceHandles(NULL)
{
        GetMTSServer()->GetSystemProperties().SetProperty("Telephony.ReleaseString",
                "GlobalCall TSP Release 0.1");
        cStopTSPEvent = ::CreateEvent(NULL, false, false, NULL);
//      deviceArrSize = 69;
//      pDeviceHandles = new long[deviceArrSize];
//      deviceCount = 0;
}
// virtual
GlobalCallTSP::~GlobalCallTSP(void)
{
        // Channels are deleted by superclass
        //
        if (gc_Stop() != GC_SUCCESS)
                error("gc_Stop() failed." );
```

```
}
// virtual
bool
GlobalCallTSP::Setup(void)
{
        /*
        * Set SRL to MultiThreaded-Async mode
        */
        int srlMode = SR_STASYNC;  // use ST because we're using our own thread for now
        if (sr_setparm( SRL_DEVICE, SR_MODELTYPE,  &srlMode) == -1 ) {
                error("Dialogic GlobalCall startup failed - Unable to set to Model Type");
                return false;
        }
        unsigned int srlSize = 10;
        if (sr_setparm( SRL_DEVICE, SR_INTERPOLLID, &srlSize) == -1 ) {
                error("Dialogic GlobalCall startup failed - Unable to set polling time");
                return false;
        }


        /*
    * GlobalCall Initialization
         * start all configured call control libraries
    */
        int ret;
        if ((ret = gc_Start(NULL)) != GC_SUCCESS) {
                error("Dialogic GlobalCall startup failed - gc_Start() returns %d.", ret);
                return false;
        }
        // No Handler -- GlobalCallTChannel::EnableHandler();
        // Create channels
        //
        int boardcnt = 0;
        if (sr_getboardcnt(DEV_CLASS_DTI, &boardcnt) != GC_SUCCESS) {
                error("Dialogic GlobalCall startup failed - sr_getboardcnt() failed.");
                return false;
        }
        if (boardcnt <= 0) {
                error("Dialogic GlobalCall startup failed - No board found or the Dialogic driver is not started.");
                return false;
        }

        bool bAllocateAll = false;
        gm::string instr, outstr;
        gm::string tspname;
        gm::string parent(MTS_REGISTRY_BASE);
        gm::string newparent;
        if (!ConfigMgr::GetString(parent, "Telephony Service Provider", tspname))
                bAllocateAll = true;
        else
        {
                newparent = parent;
                newparent += ".Telephony Service Providers.";
                newparent += tspname;
        }
        if (!bAllocateAll && !ConfigMgr::GetString(newparent, "IncomingLines", instr))
```

```
                bAllocateAll = true;
        if (!bAllocateAll && !ConfigMgr::GetString(newparent, "OutgoingLines", outstr))
                bAllocateAll = true;
        RangeParser inrange(instr);
        RangeParser outrange(outstr);
        for (int i = 0; i < boardcnt; i++)
        {
                char devname[64];
                // Create channels
                //
                sprintf(devname, "dtiB%d", i + 1);

                int bddev = dt_open(devname, 0);
                if (bddev == -1) {
                        error("Dialogic GlobalCall startup failed - Unable to open device \"%s\".", devname);
                        return false;
                }
                long subdevs = ATDV_SUBDEVS(bddev) - 1;
                if (subdevs == AT_FAILURE) {
                        error("Dialogic GlobalCall startup failed - ATDV_SUBDEVS failed, don't know how many
channels exist.");
                        return false;
                }
                char **chnamesp = ATDX_CHNAMES(bddev);
                // Allocate enough channel objects
                //
                if (!bAllocateAll)
                {
                        for (int j = 0; j < (int)subdevs; j++)
                        {
                                if (inrange.IsSet((i*subdevs)+j) || outrange.IsSet((i*subdevs)+j))
                                        NewChannel(i+1, j+1, (i*subdevs)+j);
                        }
                }
                else
                {
                        for (int j = 0; j < (int)subdevs; j++)
                                NewChannel(i+1, j+1, (i*subdevs)+j);
                }
                if (dt_close(bddev) == -1) {
                        error("Dialogic GlobalCall startup failed - Unable to close device \"%s\".", devname);
                        return false;
                }
        }
        //return go();
}
bool
GlobalCallTSP::NewChannel(int boardNum, int portNum, int id)
{
        bool ret = false;
        char buf[50];
        sprintf(buf, ":N_dtiB%dT%d:%s",
                boardNum, portNum, protocolName);
        GlobalCallTChannel *pNewChannel = new GlobalCallTChannel(this, boardNum, portNum, cLogType, id);
        if (pNewChannel && pNewChannel->Setup(buf))
        {
```

```
                info("Telephony Channel %d is up", id);
                if (!pNewChannel->IsAlive())
                {
                        int DChanStat;
                        bool goodStatus = pNewChannel->GetDChannelStatus(&DChanStat);
                        if (goodStatus && DChanStat == DATA_LINK_UP)
                                pNewChannel->SetPhysicalChannelStatus(TelephonyChannel::S_InService);
                }
                AddChannel(pNewChannel);
                ret = true;
        }
        else
        {
                delete pNewChannel;
                error("GlobalCall Telephony Channel Setup failed for device \"dtiB%dT%d\".", boardNum,
portNum);
        }
        return ret;
}
#if 0
bool
GlobalCallTSP::MonitorDevice(long deviceHandle)
{
        Synchronize sync(cCritSection);
        long *tmpArr;
        if (deviceArrSize < (deviceCount+1))
        {
                tmpArr = new long[deviceArrSize+69];
                for (int i=0; i<deviceCount; i++)
                        tmpArr[i] = pDeviceHandles[i];
                delete [] pDeviceHandles;
                pDeviceHandles = tmpArr;
        }
        pDeviceHandles[++deviceCount] = deviceHandle;
        return true;
}
#endif
bool
GlobalCallTSP::BeginService(void)
{
        TelephonyServiceProvider::BeginService();
        return go();
}
bool
GlobalCallTSP::go(void)
{
        cThreadH =
                (HANDLE)_beginthreadex( NULL,
                0,
                (PBEGINTHREADEX_THREADFUNC) _GoThreadFunc,
                (LPVOID) this,
                0,
                (PBEGINTHREADEX_THREADID) &cThreadId );
        if (cThreadH) {
                info("GlobalCallTSP thread ID %ld started", cThreadId);
                return true;
```

```
        }
        else {
                error("Could not start GlobalCallTSP thread");
                return false;
        }
}
unsigned __stdcall
GlobalCallTSP::_GoThreadFunc(LPVOID param)
{
        GlobalCallTSP *pTSP = (GlobalCallTSP *)param;
        return pTSP->GoThreadFunc();
}
unsigned
GlobalCallTSP::GoThreadFunc(void)
{
        while (true) {
                if (::WaitForSingleObject(cStopTSPEvent, 0) != WAIT_TIMEOUT) { // signaled
                        info("GlobalCallTSP is requested to stop -- exiting thread ...");
                        break;
                }
    debug("Getting Dialogic event.");
                long evtHandle = sr_waitevt(500);
                //long evtHandle;
                //long ret = sr_waitevtEx(pDeviceHandles, deviceCount, 500, &evtHandle);
                if (evtHandle != SR_TMOUT) {
                //if (ret != -1) {
      long chEventType = sr_getevttype(0);
      //long chEventType = sr_getevttype(evtHandle);
      debug("Got Dialogic event %ld.", chEventType);
      if ((chEventType >= TDX_PLAY) &&
              (chEventType <= TDX_NOSTOP)) {
        // call mediachannel
        DialogicMChannel::dx_handler(evtHandle);
      }
      else GlobalCallTChannel::dt_handler(0);  // pass 0 because sr_waitevt is used
      //else GlobalCallTChannel::dt_handler(evtHandle);
                }
    else
      debug("Got Dialogic event - timed out.");
       }
       return 0;
}
// virtual
bool
GlobalCallTSP::Teardown(void)
{
        // Channels are deleted by superclass
        //
        bool ret = true;
        std::list<Channel *>::iterator iter = cChannels.begin();
        while (iter != cChannels.end())
        {
                if (!((GlobalCallTChannel *)(*iter))->Shutdown()) {
                        ret = false;
                        error("Shutdown failed for Channel Number %ld.", ((GlobalCallTChannel *)(*iter))-
>GetHdl());
```

```
                }
                iter++;
        }
        ::SetEvent(cStopTSPEvent); // exit thread anyway
        if (cThreadH) {
                ::WaitForSingleObject(cThreadH, INFINITE);
                if (::CloseHandle(cThreadH))
                        diag("Closing handle \"cThreadH\" was successful.");
                else
                        diag("Closing handle \"cThreadH\" failed.");
        }
        return ret;
}
GlobalCallTChannel *
GlobalCallTSP::FindChannelForDescriptor(long desc)
{
        std::list<Channel *>::iterator iter = cChannels.begin();

        while ((iter != cChannels.end()) &&
                ((GlobalCallTChannel *)(*iter))->GetHdl() != desc)
                iter++;
        return (iter != cChannels.end()) ? (GlobalCallTChannel *)(*iter) : 0;
}
void
GlobalCallTSP::SetInstance(GlobalCallTSP *ptr)
{
        cInstance = ptr;
}
GlobalCallTSP *
GlobalCallTSP::GetInstance(void)
{
        return cInstance;
}
TelephonyServiceProvider *
CreateTelephonyServiceProvider(MTSServer *pServer)
{
        GlobalCallTSP::SetInstance(new GlobalCallTSP(pServer));
        return GlobalCallTSP::GetInstance();
}
bool
GlobalCallTSP::Restart(void)
{
        info("Restarting GlobalCallTSP begins now." );
        info("Bringing all the channels down." );
        BringAllChannelsDown();
        info("Dropping all the calls and cleaning up resources." );
        CleanupAllResources();
        info("Stopping GlobalCallTSP thread." );
        Teardown();
        info("Destroying all the channel objects." );
        std::list<Channel *>::iterator iter = cChannels.begin();
        while (iter != cChannels.end())
        {
                delete *iter;
                iter++;
        }
```

```
	cChannels.clear();
	info("Stopping GlobalCall Library." );
	if (gc_Stop() != GC_SUCCESS)
		error("gc_Stop() failed." );
	info("GlobalCallTSP is now restarting with new channel objects..." );
	gm::string tspname, tsplibname;
	gm::string parent(MTS_REGISTRY_BASE);
	gm::string newparent;
	if (!ConfigMgr::GetString(parent, "Telephony Service Provider", tspname)) {
		error("During TSP Restart - Problem getting \"Telephony Service Provider\" parameters from the
registry." );
		return false;
	}
	newparent = parent;
	newparent += ".Telephony Service Providers.";
	newparent += tspname;
	if (!ConfigMgr::GetString(newparent, "Library", tsplibname)) {
		error("During TSP Restart - Problem getting \"Library\" parameters from the registry." );
		return false;
	}
	if (!Setup()) {
		error("During TSP Restart - Problem in TSP Setup." );
		return false;
	}
	if (!PostSetup(newparent)) {
		error("During TSP Restart - Problem in TSP PostSetup." );
		return false;
	}
	else
		return true;
}
void
GlobalCallTSP::BringAllChannelsDown(void)
{
	std::list<Channel *>::iterator iter = cChannels.begin();
	while (iter != cChannels.end()) {
		((GlobalCallTChannel *)(*iter))->SetStatus(TelephonyChannel::S_OutOfService);
		iter++;
	}
}
GCTSPLog GCTSPLog::log;
```

```
/*
 * GMLangMgr.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:22:17 $
 */
static const char* id="$Id: GMLangMgr.cpp,v 1.2 2000/12/07 03:22:17 Mitsuru Oshima Exp $";
#include <string.h>
#include "GMLangMgr.h"
typedef struct
{
        GMLANGID langid;
        char *str;
} LANGINFO, *PLANGINFO;
static LANGINFO primaryinfo[] = {
        {GMLANG_AFRIKAANS, "af"},
        {GMLANG_ALBANIAN, "sq"},
        {GMLANG_ARABIC, "ar"},
        {GMLANG_BASQUE, "eu"},
        {GMLANG_BELARUSIAN, "be"},
        {GMLANG_BULGARIAN, "bg"},
        {GMLANG_CATALAN, "ca"},
        {GMLANG_CHINESE, "zh"},
        {GMLANG_CROATIAN, "hr"},
        {GMLANG_CZECH, "cs"},
        {GMLANG_DANISH, "da"},
        {GMLANG_DUTCH, "nl"},
        {GMLANG_ENGLISH, "en"},
        {GMLANG_ESTONIAN, "et"},
        {GMLANG_FAEROESE, "fo"},
        {GMLANG_FARSI, "fa"},
        {GMLANG_FINNISH, "fi"},
        {GMLANG_FRENCH, "fr"},
        {GMLANG_GERMAN, "de"},
        {GMLANG_GREEK, "el"},
        {GMLANG_HEBREW, "he"},
        {GMLANG_HUNGARIAN, "hu"},
        {GMLANG_ICELANDIC, "is"},
        {GMLANG_INDONESIAN, "in"},
        {GMLANG_ITALIAN, "it"},
        {GMLANG_JAPANESE, "ja"},
        {GMLANG_KOREAN, "ko"},
        {GMLANG_LATVIAN, "lv"},
        {GMLANG_LITHUANIAN, "lt"},
        {GMLANG_NORWEGIAN, "no"},
        {GMLANG_POLISH, "pl"},
        {GMLANG_PORTUGUESE, "pt"},
        {GMLANG_ROMANIAN, "ro"},
        {GMLANG_RUSSIAN, "ru"},
        {GMLANG_SERBIAN, "sr"},
```

```
        {GMLANG_SLOVAK, "sk"},
        {GMLANG_SLOVENIAN, "sl"},
        {GMLANG_SPANISH, "es"},
        {GMLANG_SWEDISH, "sv"},
        {GMLANG_THAI, "th"},
        {GMLANG_TURKISH, "tr"},
        {GMLANG_UKRAINIAN, "uk"},
        {GMLANG_VIETNAMESE, "vi"},
        {GMLANG_INVALID, ""}
};
static LANGINFO subinfo[] = {
        {MAKEGMLANGID(GMLANG_ARABIC, GMSUBLANG_ARABIC_SAUDI_ARABIA), "sa"},
        {MAKEGMLANGID(GMLANG_ARABIC, GMSUBLANG_ARABIC_IRAQ), "iq"},
        {MAKEGMLANGID(GMLANG_ARABIC, GMSUBLANG_ARABIC_EGYPT), "eg"},
        {MAKEGMLANGID(GMLANG_ARABIC, GMSUBLANG_ARABIC_LIBYA), "ly"},
        {MAKEGMLANGID(GMLANG_ARABIC, GMSUBLANG_ARABIC_ALGERIA), "dz"},
        {MAKEGMLANGID(GMLANG_ARABIC, GMSUBLANG_ARABIC_MOROCCO), "ma"},
        {MAKEGMLANGID(GMLANG_ARABIC, GMSUBLANG_ARABIC_TUNISIA), "tn"},
        {MAKEGMLANGID(GMLANG_ARABIC, GMSUBLANG_ARABIC_OMAN), "om"},
        {MAKEGMLANGID(GMLANG_ARABIC, GMSUBLANG_ARABIC_YEMEN), "ye"},
        {MAKEGMLANGID(GMLANG_ARABIC, GMSUBLANG_ARABIC_SYRIA), "sy"},
        {MAKEGMLANGID(GMLANG_ARABIC, GMSUBLANG_ARABIC_JORDAN), "jo"},
        {MAKEGMLANGID(GMLANG_ARABIC, GMSUBLANG_ARABIC_LEBANON), "lb"},
        {MAKEGMLANGID(GMLANG_ARABIC, GMSUBLANG_ARABIC_KUWAIT), "kw"},
        {MAKEGMLANGID(GMLANG_ARABIC, GMSUBLANG_ARABIC_UAE), "ae"},
        {MAKEGMLANGID(GMLANG_ARABIC, GMSUBLANG_ARABIC_BAHRAIN), "bh"},
        {MAKEGMLANGID(GMLANG_ARABIC, GMSUBLANG_ARABIC_QATAR), "qa"},
        {MAKEGMLANGID(GMLANG_CHINESE, GMSUBLANG_CHINESE_TRADITIONAL), "cn"},
        {MAKEGMLANGID(GMLANG_CHINESE, GMSUBLANG_CHINESE_SIMPLIFIED), "tw"},
        {MAKEGMLANGID(GMLANG_CHINESE, GMSUBLANG_CHINESE_HONGKONG), "hk"},
        {MAKEGMLANGID(GMLANG_CHINESE, GMSUBLANG_CHINESE_SINGAPORE), "sg"},
        {MAKEGMLANGID(GMLANG_DUTCH, GMSUBLANG_DUTCH_NETHERLANDS), "nl"},
        {MAKEGMLANGID(GMLANG_DUTCH, GMSUBLANG_DUTCH_BELGIAN), "be"},
        {MAKEGMLANGID(GMLANG_ENGLISH, GMSUBLANG_ENGLISH_US), "us"},
        {MAKEGMLANGID(GMLANG_ENGLISH, GMSUBLANG_ENGLISH_UK), "gb"},
        {MAKEGMLANGID(GMLANG_ENGLISH, GMSUBLANG_ENGLISH_AUS), "au"},
        {MAKEGMLANGID(GMLANG_ENGLISH, GMSUBLANG_ENGLISH_CAN), "ca"},
        {MAKEGMLANGID(GMLANG_ENGLISH, GMSUBLANG_ENGLISH_NZ), "nz"},
        {MAKEGMLANGID(GMLANG_ENGLISH, GMSUBLANG_ENGLISH_EIRE), "ie"},
        {MAKEGMLANGID(GMLANG_ENGLISH, GMSUBLANG_ENGLISH_SOUTH_AFRICA), "za"},
        {MAKEGMLANGID(GMLANG_ENGLISH, GMSUBLANG_ENGLISH_JAMAICA), "jm"},
        {MAKEGMLANGID(GMLANG_ENGLISH, GMSUBLANG_ENGLISH_CARIBBEAN), "us"},
        {MAKEGMLANGID(GMLANG_ENGLISH, GMSUBLANG_ENGLISH_BELIZE), "bz"},
        {MAKEGMLANGID(GMLANG_ENGLISH, GMSUBLANG_ENGLISH_TRINIDAD), "tt"},
        {MAKEGMLANGID(GMLANG_FRENCH, GMSUBLANG_FRENCH_FRANCE), "fr"},
        {MAKEGMLANGID(GMLANG_FRENCH, GMSUBLANG_FRENCH_BELGIAN), "be"},
        {MAKEGMLANGID(GMLANG_FRENCH, GMSUBLANG_FRENCH_CANADIAN), "ca"},
        {MAKEGMLANGID(GMLANG_FRENCH, GMSUBLANG_FRENCH_SWISS), "ch"},
        {MAKEGMLANGID(GMLANG_FRENCH, GMSUBLANG_FRENCH_LUXEMBOURG), "lu"},
        {MAKEGMLANGID(GMLANG_GERMAN, GMSUBLANG_GERMAN_GERMANY), "de"},
        {MAKEGMLANGID(GMLANG_GERMAN, GMSUBLANG_GERMAN_SWISS), "ch"},
        {MAKEGMLANGID(GMLANG_GERMAN, GMSUBLANG_GERMAN_AUSTRIAN), "at"},
        {MAKEGMLANGID(GMLANG_GERMAN, GMSUBLANG_GERMAN_LUXEMBOURG), "lu"},
        {MAKEGMLANGID(GMLANG_GERMAN, GMSUBLANG_GERMAN_LIECHTENSTEIN), "li"},
        {MAKEGMLANGID(GMLANG_ITALIAN, GMSUBLANG_ITALIAN_ITALY), "it"},
        {MAKEGMLANGID(GMLANG_ITALIAN, GMSUBLANG_ITALIAN_SWISS), "ch"},
```

```
        {MAKEGMLANGID(GMLANG_KOREAN, GMSUBLANG_KOREAN_KOREA), "ko"},
        {MAKEGMLANGID(GMLANG_KOREAN, GMSUBLANG_KOREAN_JOHAB), "ko"},
        {MAKEGMLANGID(GMLANG_NORWEGIAN, GMSUBLANG_NORWEGIAN_BOKMAL), "no"},
        {MAKEGMLANGID(GMLANG_NORWEGIAN, GMSUBLANG_NORWEGIAN_NYNORSK), "no"},
        {MAKEGMLANGID(GMLANG_PORTUGUESE, GMSUBLANG_PORTUGUESE_PORTUGAL), "pt"},
        {MAKEGMLANGID(GMLANG_PORTUGUESE, GMSUBLANG_PORTUGUESE_BRAZILIAN), "br"},
        {MAKEGMLANGID(GMLANG_SERBIAN, GMSUBLANG_SERBIAN_LATIN), "sr"},
        {MAKEGMLANGID(GMLANG_SERBIAN, GMSUBLANG_SERBIAN_CYRILLIC), "sr"},
        {MAKEGMLANGID(GMLANG_SPANISH, GMSUBLANG_SPANISH_SPAIN), "es"},
        {MAKEGMLANGID(GMLANG_SPANISH, GMSUBLANG_SPANISH_MEXICAN), "mx"},
        {MAKEGMLANGID(GMLANG_SPANISH, GMSUBLANG_SPANISH_MODERN), "es"},
        {MAKEGMLANGID(GMLANG_SPANISH, GMSUBLANG_SPANISH_GUATEMALA), "gt"},
        {MAKEGMLANGID(GMLANG_SPANISH, GMSUBLANG_SPANISH_COSTA_RICA), "cr"},
        {MAKEGMLANGID(GMLANG_SPANISH, GMSUBLANG_SPANISH_PANAMA), "pa"},
        {MAKEGMLANGID(GMLANG_SPANISH, GMSUBLANG_SPANISH_DOMINICAN_REPUBLIC),
"do"},
        {MAKEGMLANGID(GMLANG_SPANISH, GMSUBLANG_SPANISH_VENEZUELA), "ve"},
        {MAKEGMLANGID(GMLANG_SPANISH, GMSUBLANG_SPANISH_COLOMBIA), "co"},
        {MAKEGMLANGID(GMLANG_SPANISH, GMSUBLANG_SPANISH_PERU), "pe"},
        {MAKEGMLANGID(GMLANG_SPANISH, GMSUBLANG_SPANISH_ARGENTINA), "ar"},
        {MAKEGMLANGID(GMLANG_SPANISH, GMSUBLANG_SPANISH_ECUADOR), "ec"},
        {MAKEGMLANGID(GMLANG_SPANISH, GMSUBLANG_SPANISH_CHILE), "cl"},
        {MAKEGMLANGID(GMLANG_SPANISH, GMSUBLANG_SPANISH_URUGUAY), "uy"},
        {MAKEGMLANGID(GMLANG_SPANISH, GMSUBLANG_SPANISH_PARAGUAY), "py"},
        {MAKEGMLANGID(GMLANG_SPANISH, GMSUBLANG_SPANISH_BOLIVIA), "bo"},
        {MAKEGMLANGID(GMLANG_SPANISH, GMSUBLANG_SPANISH_EL_SALVADOR), "sv"},
        {MAKEGMLANGID(GMLANG_SPANISH, GMSUBLANG_SPANISH_HONDURAS), "hn"},
        {MAKEGMLANGID(GMLANG_SPANISH, GMSUBLANG_SPANISH_NICARAGUA), "ni"},
        {MAKEGMLANGID(GMLANG_SPANISH, GMSUBLANG_SPANISH_PUERTO_RICO), "pr"},
        {MAKEGMLANGID(GMLANG_SWEDISH, GMSUBLANG_SWEDISH_SWEEDEN), "se"},
        {MAKEGMLANGID(GMLANG_SWEDISH, GMSUBLANG_SWEDISH_FINLAND), "fi"},
        {GMSUBLANG_INVALID, ""}
};
bool
GMLanguageManager::GMLANGIDToStrings(GMLANGID langid, gm::string& primary,
        gm::string& sublang)
{
        bool ret = true;
        primary.erase();
        sublang.erase();

        if (GMPRIMARYLANGID(langid) != GMLANG_INVALID)
        {
                PLANGINFO pInfo = primaryinfo;
                while (pInfo->langid != GMLANG_INVALID)
                {
                        if (GMPRIMARYLANGID(langid) == pInfo->langid)
                        {
                                primary = pInfo->str;
                                break;
                        }
                        pInfo++;
                }
                if (pInfo->langid == GMLANG_INVALID)
                        ret = false;
        }
```

```
        if (GMSUBLANGID(langid) != GMSUBLANG_INVALID)
        {
                PLANGINFO pInfo = subinfo;
                while (pInfo->langid != GMSUBLANG_INVALID)
                {
                        if (langid == pInfo->langid)
                        {
                                sublang = pInfo->str;
                                break;
                        }
                        pInfo++;
                }
                if (pInfo->langid == GMSUBLANG_INVALID)
                        ret = false;
        }
        return ret;
}
GMLANGID
GMLanguageManager::StringsToGMLANGID(const gm::string& primary,
        const gm::string& sublang)
{
        PLANGINFO pInfo;
        GMLANGID prim = GMLANG_INVALID;
        GMLANGID sub = GMSUBLANG_INVALID;
        pInfo = primaryinfo;
        while (pInfo->langid != GMLANG_INVALID)
        {
                if (primary == pInfo->str)
                {
                        prim = pInfo->langid;
                        break;
                }
                pInfo++;
        }
        pInfo = subinfo;
        while (pInfo->langid != GMSUBLANG_INVALID)
        {
                if (sublang == pInfo->str)
                {
                        sub = GMSUBLANGID(pInfo->langid);
                        break;
                }
                pInfo++;
        }
        return MAKEGMLANGID(prim, sub);
}
```

```
/*
 * HandoffRecordingMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:21:49 $
 */
static const char* id="$Id: HandoffRecordingMsg.cpp,v 1.4 2000/12/07 03:21:49 Mitsuru Oshima Exp $";
#include "HandoffRecordingMsg.h"
HandOffRecordingMessage::HandOffRecordingMessage(CallToken& token,
        RECORDINGID RecordingID, const gm::string& ServiceName, void *pData,
        unsigned long DataLength)
        : OneCallTokenMessageBase(M_CMD_HANDOFFRECORDING, token)
{
        cRecordingID = RecordingID;
        cServiceName = ServiceName.c_str();
        cData = new unsigned char [DataLength];
        memcpy(cData, pData, DataLength);
        cDataLength = DataLength;
}
HandOffRecordingMessage::~HandOffRecordingMessage(void)
{
        delete [] cData;
}
// virtual
void
HandOffRecordingMessage::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        InsertUshort((unsigned short)cRecordingID);
        InsertString(cServiceName);
        InsertUlong(cDataLength);
        InsertRaw((char *)cData, cDataLength);
}
// virtual
void
HandOffRecordingMessage::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        cRecordingID = (RECORDINGID)ExtractUshort();
        ExtractString(cServiceName);
        cDataLength = ExtractUlong();
        if (cDataLength)
        {
                cData = new unsigned char[cDataLength];
                ExtractRaw((char *)cData, cDataLength);
        }
        else
                cData = 0;
}
```

```
/*
 * IDLog.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.7 $
 * $Date: 2000/12/07 03:22:17 $
 */
static const char* id="$Id: IDLog.cpp,v 1.7 2000/12/07 03:22:17 Mitsuru Oshima Exp $";
#include <stdio.h>
#include <stdarg.h>
#include "IDLog.h"
#include "GMString"
#include "Log.h"
IDLog::IDLog(const char* name)
{
        log = & (VPPLog::getLogFor(name));
        mID.resize(4, '\0');
}
void
IDLog::setLogID(const char* string)
{
        mID = string;
}
void
IDLog::setLogID(const char* fmt, ...)
{
        va_list vl;
        va_start(vl, fmt);
  this->va_setLogID(fmt, vl);
}
void
IDLog::va_setLogID(const char* fmt, va_list vl)
{
  char buf[256];
  ::_vsnprintf(buf, sizeof(buf), fmt, vl);
  buf[sizeof(buf)-1] = '\0';
  mID = buf;
}
void
IDLog::error(const char* fmt, ...)
{
        va_list vl;
        va_start(vl, fmt);
  char buf[256];
  if (::_vsnprintf(buf, sizeof(buf) - 1, fmt, vl) <0) {
    char *buf = new char[65535];
    if (::_vsnprintf(buf, 65535, fmt, vl) <0) {
      //give up
      buf[65534]=0;
    }
    log->error("%s %s", mID.c_str(), buf);
```

```
    delete buf;
    return;
  };
        buf[sizeof(buf) - 1] = '\0';
  log->error("%s %s", mID.c_str(), buf);
}
void
IDLog::warning(const char* fmt, ...)
{
        va_list vl;
        va_start(vl, fmt);
  char buf[256];
  if (::_vsnprintf(buf, sizeof(buf) - 1, fmt, vl) <0) {
    char *buf = new char[65535];
    if (::_vsnprintf(buf, 65535, fmt, vl) <0) {
      //give up
      buf[65534]=0;
    }
    log->warning("%s %s", mID.c_str(), buf);
    delete buf;
    return;
  };
        buf[sizeof(buf) - 1] = '\0';
        log->warning("%s %s", mID.c_str(), buf);
}
void
IDLog::info(const char* fmt, ...)
{
        va_list vl;
        va_start(vl, fmt);
  char buf[256];
  if (::_vsnprintf(buf, sizeof(buf) - 1, fmt, vl) <0) {
    char *buf = new char[65535];
    if (::_vsnprintf(buf, 65535, fmt, vl) <0) {
      //give up
      buf[65534]=0;
    }
    log->info("%s %s", mID.c_str(), buf);
    delete buf;
    return;
  };
        buf[sizeof(buf) - 1] = '\0';
  log->info("%s %s", mID.c_str(), buf);
}
void
IDLog::diag(const char* fmt, ...)
{
        va_list vl;
        va_start(vl, fmt);
  char buf[256];
  if (::_vsnprintf(buf, sizeof(buf) - 1, fmt, vl) <0) {
    char *buf = new char[65535];
    if (::_vsnprintf(buf, 65535, fmt, vl) <0) {
      //give up
      buf[65534]=0;
    }
```

```
    log->diag("%s %s", mID.c_str(), buf);
    delete buf;
    return;
  };
        buf[sizeof(buf) - 1] = '\0';
  log->diag("%s %s", mID.c_str(), buf);
}
void
IDLog::debug(const char* fmt, ...)
{
        va_list vl;
        va_start(vl, fmt);
  char buf[256];
  if (::_vsnprintf(buf, sizeof(buf) - 1, fmt, vl) <0) {
    char *buf = new char[65535];
    if (::_vsnprintf(buf, 65535, fmt, vl) <0) {
      //give up
      buf[65534]=0;
    }
    log->debug("%s %s", mID.c_str(), buf);
    delete buf;
    return;
  };
        buf[sizeof(buf) - 1] = '\0';
  log->debug("%s %s", mID.c_str(), buf);
}
```

```
/*
 * IncomingCallNtfy.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:49 $
 */
static const char* id="$Id: IncomingCallNtfy.cpp,v 1.3 2000/12/07 03:21:49 Mitsuru Oshima Exp $";
#include "IncomingCallNtfy.h"
// virtual
void
IncomingCallNotification::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        cDNIS.Serialize(this);
        cANI.Serialize(this);
}
// virtual
void
IncomingCallNotification::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        cDNIS.Deserialize(this);
        cANI.Deserialize(this);
}
```

Exhibit 2009, Page 5 of 5

```
/*
 * InsertDynamicGrammarMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:49 $
 */
static const char* id="$Id: InsertDynamicGrammarMsg.cpp,v 1.2 2000/12/07 03:21:49 Mitsuru Oshima Exp $";
#include "InsertDynamicGrammarMsg.h"
// virtual
void
InsertDynamicGrammarMessage::Serialize(void)
{
        DynamicGrammarMessage::Serialize();
        InsertString(cGrammarName);
}
// virtual
void
InsertDynamicGrammarMessage::Deserialize(void)
{
        DynamicGrammarMessage::Deserialize();
        ExtractString(cGrammarName);
}
```

```
/*
 * ConnectionWatcher.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:19:46 $
 */
static const char* id="$Id: ConnectionWatcher.cpp,v 1.2 2000/12/07 03:19:46 Mitsuru Oshima Exp $";
#include "ConnectionMgr.h"
#include "ConnectionWatcher.h"
ConnectionWatcher::ConnectionWatcher(invui::Pingable_ptr p, char* _id) : Thread()
{
   ping = invui::Pingable::_duplicate(p);
   id = CORBA::strdup(_id);
}
ConnectionWatcher::~ConnectionWatcher() {
   CORBA::string_free(id);
}
void
ConnectionWatcher::run() {
   while (true) {
      try {
         ping->ping();
      } catch (const CORBA::Exception& e) {
         break;
      }
   }
   ConnectionMgr::shutdown(id);
}
```

```
/*
 * IPCEndpoint.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.6 $
 * $Date: 2000/12/07 03:20:47 $
 */
static const char* id="$Id: IPCEndpoint.cpp,v 1.6 2000/12/07 03:20:47 Mitsuru Oshima Exp $";
#include <windows.h>
#include "IPCEndpoint.h"
#include "VPPUtils/VPPDebug.h"
IPCEndpoint::IPCEndpoint(void)
{
        debug::printf("IPCEndpoint", "Creating IPCEndpoint %p\n", this);
        cUseMutexes = false;
        SetUseMutexes(true);
        cNearEndDisconnected = false;
}
// virtual
IPCEndpoint::~IPCEndpoint(void)
{
        debug::printf("IPCEndpoint", "Destroying IPCEndpoint %p\n", this);
        if (cUseMutexes)
        {
                ::CloseHandle((HANDLE)cReadMutex);
                ::CloseHandle((HANDLE)cWriteMutex);
        }
}
// virtual
bool
IPCEndpoint::Connect(void)
{
        cNearEndDisconnected = false;
        return true;
}
// virtual
bool
IPCEndpoint::Disconnect(void)
{
        cNearEndDisconnected = true;
        return true;
}
bool
IPCEndpoint::Write(unsigned char byte, unsigned short NumUserObjects,
        unsigned long *UserObjects)
{
        unsigned long BytesWritten;
        BeginAtomicWrite();
        bool ret = Write(&byte, 1, &BytesWritten, NumUserObjects, UserObjects);
        EndAtomicWrite();
        return (ret && BytesWritten == 1);
```

```
}
bool
IPCEndpoint::Read(unsigned char& byte, unsigned short NumUserObjects,
        unsigned long *UserObjects)
{
        unsigned long BytesRead;
        BeginAtomicRead();
        bool ret = Read(&byte, 1, &BytesRead, NumUserObjects, UserObjects);
        EndAtomicRead();
        return (ret && BytesRead == 1);
}
bool
IPCEndpoint::Write(unsigned short us, unsigned short NumUserObjects, unsigned long *UserObjects)
{
        unsigned char hi = (unsigned char)((us >> 8) & 0xff);
        unsigned char lo = (unsigned char)(us & 0xff);
        bool ret = false;
        BeginAtomicWrite();
        if (Write(hi, NumUserObjects, UserObjects))
                ret = Write(lo, NumUserObjects, UserObjects);
        EndAtomicWrite();
        return ret;
}
bool
IPCEndpoint::Read(unsigned short& us, unsigned short NumUserObjects, unsigned long *UserObjects)
{
        unsigned char hi, lo;
        bool ret = true;
        BeginAtomicRead();

        if (Read(hi, NumUserObjects, UserObjects) &&
                Read(lo, NumUserObjects, UserObjects))
                us = (unsigned short)((hi << 8) | lo);
        else
                ret = false;

        EndAtomicRead();
        return ret;
}
bool
IPCEndpoint::Write(unsigned long ul, unsigned short NumUserObjects, unsigned long *UserObjects)
{
        unsigned short hi = (unsigned short)((ul >> 16) & 0xffff);
        unsigned short lo = (unsigned short)(ul & 0xffff);
        bool ret = false;
        BeginAtomicWrite();

        if (Write(hi, NumUserObjects, UserObjects))
                ret = Write(lo, NumUserObjects, UserObjects);
        EndAtomicWrite();
        return ret;
}
bool
IPCEndpoint::Read(unsigned long& ul, unsigned short NumUserObjects, unsigned long *UserObjects)
{
        unsigned short hi, lo;
```

```
	bool ret = true;
	BeginAtomicRead();

	if (Read(hi, NumUserObjects, UserObjects) &&
		Read(lo, NumUserObjects, UserObjects))
		ul = (unsigned long)((hi << 16) | lo);
	else
		ret = false;
	EndAtomicRead();
	return ret;
}
void
IPCEndpoint::BeginAtomicRead(void)
{
	debug::printf("IPCEndpoint", "Thread %ld acquiring IPCEndpoint read mutex\n",
		::GetCurrentThreadId());
	if (cUseMutexes)
		::WaitForSingleObject((HANDLE)cReadMutex, INFINITE);
	debug::printf("IPCEndpoint", "Thread %ld acquired IPCEndpoint read mutex\n",
		::GetCurrentThreadId());
}
void
IPCEndpoint::EndAtomicRead(void)
{
	debug::printf("IPCEndpoint", "Thread %ld releasing IPCEndpoint read mutex\n",
		::GetCurrentThreadId());
	if (cUseMutexes)
		::ReleaseMutex((HANDLE)cReadMutex);
	debug::printf("IPCEndpoint", "Thread %ld released IPCEndpoint read mutex\n",
		::GetCurrentThreadId());
}
void
IPCEndpoint::BeginAtomicWrite(void)
{
	debug::printf("IPCEndpoint", "Thread %ld acquiring IPCEndpoint write mutex\n",
		::GetCurrentThreadId());
	if (cUseMutexes)
		::WaitForSingleObject((HANDLE)cWriteMutex, INFINITE);
	debug::printf("IPCEndpoint", "Thread %ld acquired IPCEndpoint write mutex\n",
		::GetCurrentThreadId());
}
void
IPCEndpoint::EndAtomicWrite(void)
{
	debug::printf("IPCEndpoint", "Thread %ld releasing IPCEndpoint write mutex\n",
		::GetCurrentThreadId());
	if (cUseMutexes)
		::ReleaseMutex((HANDLE)cWriteMutex);
	debug::printf("IPCEndpoint", "Thread %ld released IPCEndpoint write mutex\n",
		::GetCurrentThreadId());
}
void
IPCEndpoint::SetUseMutexes(bool flag)
{
	if (flag && !cUseMutexes)
	{
```

```
            cReadMutex = (unsigned long)::CreateMutex(NULL, FALSE, NULL);
            cWriteMutex = (unsigned long)::CreateMutex(NULL, FALSE, NULL);
        }
        else if (!flag && cUseMutexes)
        {
            ::CloseHandle((HANDLE)cReadMutex);
            cReadMutex = (unsigned long)INVALID_HANDLE_VALUE;
            ::CloseHandle((HANDLE)cWriteMutex);
            cWriteMutex = (unsigned long)INVALID_HANDLE_VALUE;
        }
        cUseMutexes = flag;
}
```

Document not found

```
/*
 * LogItemMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:50 $
 */
static const char* id="$Id: LogItemMsg.cpp,v 1.2 2000/12/07 03:21:50 Mitsuru Oshima Exp $";
#include "LogItemMsg.h"
// virtual
void
LogItemMessage::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        InsertString(cItemName);
        InsertString(cItemValue);
}
// virtual
void
LogItemMessage::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        ExtractString(cItemName);
        ExtractString(cItemValue);
}
```

```
/*
 * LogEnterStateMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:50 $
 */
static const char* id="$Id: LogEnterStateMsg.cpp,v 1.2 2000/12/07 03:21:50 Mitsuru Oshima Exp $";
#include "LogEnterStateMsg.h"
// virtual
void
LogEnterStateMessage::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        InsertString(cStateName);
}
// virtual
void
LogEnterStateMessage::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        ExtractString(cStateName);
}
```

Confidential

```
/*
 * LogGroups.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:22:18 $
 */
static const char* id="$Id: LogGroups.cpp,v 1.5 2000/12/07 03:22:18 Mitsuru Oshima Exp $";
#include "LogGroups.h"
#include "DefaultLog.h"
//
UtilsLog UtilsLog::log;
AutoClientLog AutoClientLog::log;
```

Document ID: 2000-0000-0000

```
/*
 * Log.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.7 $
 * $Date: 2000/12/07 03:22:18 $
 */
static const char* id="$Id: Log.cpp,v 1.7 2000/12/07 03:22:18 Mitsuru Oshima Exp $";
#include <string.h>
#include "Log.h"
#include "DefaultLog.h"
#include "Synchronized.h"
VPPLog&
VPPLog::getLogFor(const char* name) {
  Synchronized sync(log_table);
  if (log_table->containsKey((char*)name) == false) {
    if (factory == NULL) {
      factory = new DefaultLogFactory();
    }
    VPPLog* log = factory->createLog(name);
    log_table->put((char*)name, log);
  }
  return (* (log_table->get((char*)name)));
}
LogFactory*
VPPLog::setLogFactory(LogFactory* fact) {
  LogFactory* old = factory;
  factory = fact;
  return old;
}
LogFactory* VPPLog::factory = NULL;
class _MyHash : public Hashtable<char*, VPPLog*> {
public:
  _MyHash() : Hashtable<char*, VPPLog*>() { };
  ~_MyHash() {
    clear();
  };
  int _hash(char* value) {
    return strlen(value) + value[0];
  }
  char* _copy(char* value) {
    char* copy = new char[strlen(value)+1];
    strcpy(copy, value);
    return copy;
  }
  bool _keyEquals(char* k1, char* k2) {
    return strcmp(k1, k2) == 0;
  }
  bool _valueEquals(VPPLog* v1, VPPLog* v2) {
    return v1 == v2;
  }
```

```
  void _deleteKey(char* key) {
    delete [] key;
  }
};

Hashtable<char*, VPPLog*>* VPPLog::log_table = new _MyHash();
char*
Hashtable<char*, VPPLog*>::NULL_KEY = NULL;
VPPLog*
Hashtable<char*, VPPLog*>::NULL_VALUE = NULL;
```

```
/*
 * MediaChannel.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.97 $
 * $Date: 2000/12/07 03:19:48 $
 */
static const char* id="$Id: MediaChannel.cpp,v 1.97 2000/12/07 03:19:48 Mitsuru Oshima Exp $";
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <io.h>
#include "MTSServer.h"
#if 0
#include "MTSServerNode.h"
#endif
#include "MediaChannel.h"
#include "MediaServiceProvider.h"
#include "TelephonyServiceProvider.h"
#include "PromptMgr.h"
#include "SampleSegment.h"
#include "SilenceSegment.h"
#include "DTMFSegment.h"
#include "TTSSegment.h"
#include "SpeechChannel.h"
#include "TTSChannel.h"
#include "WFStreamer.h"
#include "VPPMsg/Property.h"
#include "VPPMsg/PlayMsg.h"
#include "VPPMsg/RecordMsg.h"
#include "VPPMsg/TTSMsg.h"
#include "VPPMsg/CreateCompositePromptMsg.h"
#include "VPPMsg/AppendCompositePromptPromptMsg.h"
#include "VPPMsg/AppendCompositePromptSilenceMsg.h"
#include "VPPMsg/AppendCompositePromptDTMFMsg.h"
#include "VPPMsg/AppendCompositePromptTTSMsg.h"
#include "VPPMsg/EndCompositePromptMsg.h"
#include "VPPMsg/PlayCompositePromptMsg.h"
#include "VPPMsg/RemoveRecordingMsg.h"
#include "VPPUtils/Synchronize.h"
#include "VPPUtils/GMString"
#include "VPPUtils/ConfigMgr.h"
#include "SessionNode.h"
// virtual
MediaChannel::~MediaChannel(void)
{
        ::CloseHandle(cEndOfStreamPlayDataEvent);
        ::CloseHandle(cStreamingWriteMutex);
}
void
```

```
MediaChannel::CleanupRecordings(void)
{
        gm::string str;
        while (cRecordingMgr.GetFilename(0, str))
        {
                _unlink(str);
                cRecordingMgr.Remove(str);
        }
}
unsigned long
MediaChannel::DoPlay(CallToken& ct,
        const gm::string& filename, bool isTTS, bool useCallback)
{
        unsigned long ret = MSPF_OK;
        if (useCallback && !cEndOfStreamPlayDataEvent)
        {
                ret = MSPF_CANTPLAY;
         error("StreamReadEvent does not exist - This MediaChannel Object is not in a good state");
        }
        if (ret == MSPF_OK)
                ret = Play(filename, isTTS, useCallback);
        if (ret == MSPF_OK)
        {
                if (isTTS)
                        SetState(S_PlayingTTS);
                else
                        SetState(S_Playing);
                cPlayingFilename = filename;
#ifdef NOTIFY_ALL
                if (isTTS)
        cSession->NotifyPlayingTTS(cCurrentCallToken);
                else
        cSession->NotifyPlaying(cCurrentCallToken);
#endif
        }
        return ret;
}
unsigned long
MediaChannel::DoPlaySilence(CallToken& ct, unsigned long durationmsecs)
{
        unsigned long ret = MSPF_OK;
        ret = PlaySilence(durationmsecs);
        if (ret == MSPF_OK) {
                SetState(S_Playing);
#ifdef NOTIFY_ALL
    cSession->NotifyPlaying(cCurrentCallToken);
#endif
        }
        return ret;
}
unsigned long
MediaChannel::DoPlayDTMF(CallToken& ct, const gm::string& DTMFString)
{
  unsigned long ret = MSPF_OK;
  ret = PlayDTMF(DTMFString);
        if (ret == MSPF_OK) {
```

```
                SetState(S_Playing);
#ifdef NOTIFY_ALL
    cSession->NotifyPlaying(cCurrentCallToken);
#endif
        }
  return ret;
}
bool
MediaChannel::HandlePlay(SessionNode *pSession,
        PlayMessage *pMsg, MediaChannel *pRecordChannel,
        void *userdata)
{
        if (!VerifySession(pSession))
                return false;
        unsigned long ret = MSPF_OK;

        ChannelState state = GetState();
        if (state == S_Idle)
        {
                gm::string filename;
                if (pMsg->GetRecordingID() != INVALID_RECORDINGID)
                {
                        if (!pRecordChannel->cRecordingMgr.Lookup(pMsg->GetRecordingID(), filename))
                        {
                error("no such recorded file");
                                ret = MSPF_NOSUCHRECORDING;
                                filename.erase();
                        }
                }
                else if (pMsg->GetLangID() == INVALID_GMLANGID)
                {
                        // Play from a fullpath
                        //
                        if (!pMsg->GetUseSearchPath())
                                filename = pMsg->GetPromptName();
                        else
                        {
                                // Play from a search path
                                //
                                gm::string promptSearchPath;
                                cSession->cProperties.GetProperty("Media.PromptSearchPath",
promptSearchPath);
                                if (!PromptManager::BuildPromptFilenameUsingSearchPath(promptSearchPath,
                                        pMsg->GetPromptName(), filename))
                                {
                                        warning("Couldn't find prompt file using search path: %s",
promptSearchPath.c_str());
                                        ret = MSPF_CANTFINDPROMPT;
                                        filename.erase();
                                }
                        }
                }
                else if (!PromptManager::BuildPromptFilename(pMsg->GetProductName(),
                        pMsg->GetPromptName(), pMsg->GetLangID(), filename))
                {
                        error("Couldn't build prompt filename");
```

```
                ret = MSPF_CANTPLAY;
                filename.erase();
            }
            if (ret == MSPF_OK)
                ret = DoPlay(pMsg->GetCallToken(), filename, false, false);
            const char *alternateTTS = pMsg->GetTTSAlternate().c_str();
            if (ret == MSPF_CANTFINDPROMPT && alternateTTS && strcmp(alternateTTS, ""))
            {
                debug("Playing alternate TTS");
                // play the alternative tts
                //
                ret = DoPlayTTS(pMsg->GetCallToken(), pMsg->GetTTSAlternate());
            }

            if (ret == MSPF_OK)
            {
                if (pSpChanTBNotified)
                    pSpChanTBNotified->NotifyPlayStarted();
                return true;
            }
        }
        else {
        error("HandlePlay: MedialChannel is not idle %d", GetState());
            ret = MSPF_NOTIDLE;
    }
        pSession->NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), ret);
  return false;
}
// virtual
bool
MediaChannel::HandleRecord(SessionNode *pSession, RecordMessage *pMsg, void *userdata)
{
        if (!VerifySession(pSession))
            return false;
        unsigned long ret = MSPF_OK;
        ChannelState state = GetState();
        if (state == S_Idle)
        {
            gm::string filename = pMsg->GetFileName();
            if (!filename.length())
            {
                if (!ConfigMgr::GetString(MTS_REGISTRY_BASE, "DefaultRecordingsDir", filename) ||
                    (filename.c_str() && !strcmp(filename.c_str(), "")))
                    filename = ".\\";
                if (filename.c_str() &&
                    ((*(filename.c_str()+strlen(filename.c_str())-1)) != '/') &&
                    ((*(filename.c_str()+strlen(filename.c_str())-1)) != '\\')
                    )
                    filename += "/";
                if (PromptManager::GenerateRandomUniqueFilename(filename))
                {
                    info("Filename generated for this recording: %s", filename.c_str());
                }
                else
                {
                    error("Couldn't build recording filename");
```

```
                        ret = MSPF_CANTRECORD;
                        filename.erase();
                    }
                }
                else
                {
                    // use the given filename
                    info("Given filename for this recording: %s", filename.c_str());
                }
/*              if (!PromptManager::BuildRecordingFilename(pMsg->GetProductName(),
                    pMsg->GetLangID(), filename))
                {
                    error("Couldn't build recording filename");
                    ret = MSPF_CANTRECORD;
                    filename.erase();
                }
*/
                if (ret == MSPF_OK)
                    ret = Record(filename, false, pMsg->GetBosTimeout(), pMsg->GetTmsTimeout(), pMsg->GetEosTimeout());
                if (ret == MSPF_OK)
                {
                    SetState(S_Recording);

                    RECORDINGID recordingID = cRecordingMgr.Add(filename);
                    cSession->NotifyRecording(cCurrentCallToken,
                        recordingID, filename);

                    return true;
                }
            }
            else {
                error("HandleRecord: MedialChannel is not idle %d", GetState());
                ret = MSPF_NOTIDLE;
        }
            pSession->NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), ret);
            return false;
}
unsigned long
MediaChannel::DoStop(void)
{
        unsigned long ret;

        ChannelState OldState = GetState();
        if (cPlayingComposite && (OldState == S_Playing || OldState == S_PlayingTTS))
        {
            SessionNode *pSession = cSession;
            if (pSession)
            {
                Synchronized sync(pSession);
                long timeLeft = 5000;
                long targetTime = timeLeft + ::GetTickCount();
                while (!cCompositePlayStopped && timeLeft > 0)
                {
                    sync.wait(timeLeft);
                    timeLeft = targetTime - ::GetTickCount();
```

```
                        }
                        if (!cCompositePlayStopped)
                        {
                                error("This should not happen!!! Cannot stop playing composite - previous
stopping is not done yet");
                        }
                        cStoppingCompositePlay = true;
                        cCompositePlayStopped = false;
                }
                debug("DoStop() - set cStoppingCompositePlay to true and cCompositePlayStopped to false");
        }
        if (OldState == S_Playing || OldState == S_PlayingTTS || OldState == S_Recording)
        {
                gm::string playingFileName = cPlayingFilename; // save the file name before it gets wiped out by
NotifyStopped()
                if (OldState == S_Recording)
                        SetState(S_StopRecordRequested);
                else
                        SetState(S_StopPlayRequested);
                if (OldState == S_PlayingTTS)
                {
      debug("In DoStop(), Stopping TTS");
                        StopTTS();
      debug("In DoStop(), Stopped TTS");
                }
    debug("In DoStop(), calling StopAndNotify()");
                ret = StopAndNotify();
    debug("In DoStop(), called StopAndNotify()");
        }
        else {
      error("nothing to stop");
                ret = MSPF_NOTHINGTOSTOP;
    }
        return ret;
}
bool
MediaChannel::StopCompositePlay(void)
{
 ChannelState OldState = GetState();
 if (cPlayingComposite && (OldState == S_Playing || OldState == S_PlayingTTS))
 {
  cPromptSegmentMgr.GetLast();
  cStoppingCompositePlay = true;
  return true;
 }
 return false;
}
//virtual
bool
MediaChannel::HandleStop(SessionNode *pSession, OneCallTokenMessageBase *pMsg, void *userdata)
{
        if (!VerifySession(pSession))
                return false;
        unsigned long ret = DoStop();

        if (ret != MSPF_OK)
```

```
        {
                pSession->NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), ret);
        }
        return (bool)(ret == MSPF_OK);
}
// virtual
bool
MediaChannel::HandleTTS(SessionNode *pSession, TTSMessage *pMsg, void *userdata)
{
        if (!VerifySession(pSession))
                return false;
        unsigned long ret = MSPF_OK;
        if (pMsg) {
                cMsgSerialNumber = pMsg->GetSerialNo();
                if (GetState() == S_Idle)
                {
                        if (ret == MSPF_OK)
                                ret = DoPlayTTS(pMsg->GetCallToken(), pMsg->GetText());
                }
                else {
            error("HandleTTS: MediaChannel is not idle");
                        ret = MSPF_NOTIDLE;
         }
        }
        else {
       error("null message?");
                ret = MSPF_CANTTTS;
        }
        if (ret == MSPF_OK)
                return true;
        else
        {
                pSession->NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), ret);
    return false;
        }
}
// virtual
bool
MediaChannel::HandleCreateCompositePrompt(SessionNode *pSession, CreateCompositePromptMessage *pMsg,
void *userdata)
{
        if (!VerifySession(pSession))
                return false;
        unsigned long ret = MSPF_OK;

        if (GetState() == S_Idle)
        {
#ifdef NOTIFY_ALL
      cSession->NotifyCompositePromptCreated(pMsg->GetCallToken());
#endif
                debug("Notified composite prompt created");
                cPromptSegmentMgr.Clear();
                return true;
        }
        else {
       error("HandleCreateCompositePrompt: MedialChannel is not idle %d", GetState());
```

```
                ret = MSPF_NOTIDLE;
        }
        pSession->NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), ret);
        return false;
}
// virtual
bool
MediaChannel::HandleAppendCompositePromptPrompt(SessionNode *pSession,
AppendCompositePromptPromptMessage *pMsg, void *userdata)
{
        if (!VerifySession(pSession))
                return false;
        unsigned long ret = MSPF_OK;
        bool bUseSearchPath = pMsg->GetUseSearchPath();

        if (GetState() == S_Idle)
        {
                gm::string filename;
                ret = MSPF_OK;

                if (pMsg->GetLangID() == INVALID_GMLANGID)
                {
                        // Play from a fullpath
                        //
                        if (!bUseSearchPath)
                                filename = pMsg->GetPromptName();
                        else
                        {
                                // Play from a search path
                                //
                                gm::string promptSearchPath;
                                cSession->cProperties.GetProperty("Media.PromptSearchPath",
promptSearchPath);
                                if (!PromptManager::BuildPromptFilenameUsingSearchPath(promptSearchPath,
                                        pMsg->GetPromptName(), filename))
                                {
                                        warning("Couldn't find prompt file %s using search path: %s", pMsg-
>GetPromptName().c_str(), promptSearchPath.c_str());
                                        ret = MSPF_CANTFINDPROMPT;
                                        filename.erase();
                                }
                                else
                                        debug("found %s using search path: %s", filename.c_str(),
promptSearchPath.c_str());
                        }
                }
                else if (!PromptManager::BuildPromptFilename(pMsg->GetProductName(),
                        pMsg->GetPromptName(), pMsg->GetLangID(), filename))
                {
                        error("Couldn't build prompt filename");
                        ret = MSPF_CANTPLAY;
                        filename.erase();
                }
                if ((ret == MSPF_OK) || bUseSearchPath)
                {
                        int retFileAccess = 0;
```

```
                        if ((bUseSearchPath && (ret != MSPF_OK))
                                || (!bUseSearchPath && (retFileAccess = _access(filename, 0)) != 0))
                        {
                                info("File %s cannot be accessed. Return Code = %d", (char *)(const char
*)filename, bUseSearchPath ? 1 : retFileAccess);
                                const char* alt = pMsg->GetTTSAlternate().c_str();
                                if (alt == NULL || strlen(alt) == 0)
                                {
                                        error("Couldn't find prompt %s", pMsg->GetPromptName().c_str());
                                        ret = MSPF_CANTFINDPROMPT;
                                }
                                else
                                {
                                        info("Appending TTS \"%s\" because I can\'t find the prompt", pMsg-
>GetTTSAlternate().c_str());
                                        ret = DoAppendCompositePromptTTS(pMsg->GetCallToken(),
                                                pMsg->GetProductName(), pMsg->GetLangID(), pMsg-
>GetTTSAlternate());
                                        if (ret == MSPF_OK)
                                                return true;
                                }
                        }
                        else
                        {
#ifdef NOTIFY_ALL
                cSession->NotifyCompositePromptAppended(pMsg->GetCallToken());
#endif
                                cPromptSegmentMgr.AddSegment(
                                        new SampleSegment(filename, false));
                                return true;
                        }
                }
        }
        else {
       error("HandleAppendCompositePromptPrompt: MedialChannel is not idle %d", GetState());
                ret = MSPF_NOTIDLE;
    }
        pSession->NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), ret);
  return false;
}
// virtual
bool
MediaChannel::HandleAppendCompositePromptSilence(SessionNode *pSession,
AppendCompositePromptSilenceMessage *pMsg, void *userdata)
{
        if (!VerifySession(pSession))
                return false;
        unsigned long ret = MSPF_OK;

        if (GetState() == S_Idle)
        {
                PromptSegment *pSeg = cPromptSegmentMgr.GetLast();
                if (pSeg && pSeg->GetSegmentType() == PromptSegment::ST_Silence)
                        ((SilenceSegment *)pSeg)->Append(pMsg->GetDuration());
                else
                        cPromptSegmentMgr.AddSegment(new SilenceSegment(pMsg->GetDuration()));
```

```
#ifdef NOTIFY_ALL
                cSession->NotifyCompositePromptAppended(pMsg->GetCallToken());
#endif
                return true;
        }
        else {
      error("HandleAppendCompositePromptSilence: MedialChannel is not idle %d", GetState());
                ret = MSPF_NOTIDLE;
    }
        pSession->NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), ret);
        return false;
}
// virtual
bool
MediaChannel::HandleAppendCompositePromptDTMF(SessionNode *pSession,
AppendCompositePromptDTMFMessage *pMsg, void *userdata)
{
        if (!VerifySession(pSession))
                return false;
        unsigned long ret = MSPF_OK;

        if (GetState() == S_Idle)
                cPromptSegmentMgr.AddSegment(new DTMFSegment(pMsg->GetDTMFString()));
        else {
      error("HandleAppendCompositePromptDTMF: MedialChannel is not idle %d", GetState());
                ret = MSPF_NOTIDLE;
    }
        if (ret != MSPF_OK)
        {
                pSession->NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), ret);
                return false;
        }
        return true;
}
#ifdef LATER
// virtual
bool
MediaChannel::HandleAppendCompositePromptTime(MTSServerNode *pServerNode,
        ClientID& VUIClientID, AppendCompositePromptTimeMessage *pMsg,
        void *userdata)
{

        GetMSP()->NotifyCompositePromptAppended(VUIClientID,
                pMsg->GetCallToken());
}
// virtual
bool
MediaChannel::HandleAppendCompositePromptDate(MTSServerNode *pServerNode,
        ClientID& VUIClientID, AppendCompositePromptDateMessage *pMsg,
        void *userdata)
{
        GetMSP()->NotifyCompositePromptAppended(VUIClientID,
                pMsg->GetCallToken());
}
// virtual
bool
```

```
MediaChannel::HandleAppendCompositePromptNumber(MTSServerNode *pServerNode,
        ClientID& VUIClientID, AppendCompositePromptNumberMessage *pMsg,
        void *userdata)
{
        GetMSP()->NotifyCompositePromptAppended(VUIClientID,
                pMsg->GetCallToken());
}
// virtual
bool
MediaChannel::HandleAppendCompositePromptPhoneNumber(MTSServerNode *pServerNode,
        ClientID& VUIClientID, AppendCompositePromptPhoneNumberMessage *pMsg,
        void *userdata)
{
        GetMSP()->NotifyCompositePromptAppended(VUIClientID,
                pMsg->GetCallToken());
}
#endif
unsigned long
MediaChannel::DoAppendCompositePromptTTS(CallToken& ct,
        const gm::string& product, GMLANGID lang, const gm::string& tts)
{
   gm::string filename;
   unsigned long ret;
   gm::string text = tts;
   text += " "; // to avoid words being concatinated.
#if 0
        if (PromptManager::BuildTTSFilename(product, lang, filename))
#endif
        if (PromptManager::BuildTTSFilename(filename))
        {
                PromptSegment *pSeg = cPromptSegmentMgr.GetLast();
                if (pSeg && pSeg->GetSegmentType() == PromptSegment::ST_TTS)
                        ((TTSSegment *)pSeg)->Append(text);
                else
                        cPromptSegmentMgr.AddSegment(new TTSSegment(text, filename));
#ifdef NOTIFY_ALL
                cSession->NotifyCompositePromptAppended(ct);
#endif
                ret = MSPF_OK;
        }
        else
        {
                ret = MSPF_CANTPLAY;
                error("Couldn't build prompt filename");
        }
        return ret;
}

// virtual
bool
MediaChannel::HandleAppendCompositePromptTTS(SessionNode *pSession, AppendCompositePromptTTSMessage
*pMsg, void *userdata)
{
        if (!VerifySession(pSession))
                return false;
        unsigned long ret = MSPF_OK;
```

```
        if (GetState() == S_Idle)
                ret = DoAppendCompositePromptTTS(pMsg->GetCallToken(),
                        pMsg->GetProductName(), pMsg->GetLangID(), pMsg->GetText());
        else {
       error("HandleAppendCompositePromptTTS: MedialChannel is not idle %d", GetState());
                ret = MSPF_NOTIDLE;
    }
        if (ret != MSPF_OK)
        {
                pSession->NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), ret);
                return false;
        }
        return true;
}
unsigned long
MediaChannel::DoPlayTTS(CallToken& ct, const gm::string& ttsText)
{
        unsigned long ret = MSPF_OK;
        bool bFileCleanUp = true;
        gm::string      nullfile = "";
        if (cSession->IsGoingAway())
                return MSPF_CANTTTS;
        if (!cSession->cTTSChannel)     // Allocate TTS Channel
                return MSPF_INSUFFICIENTRESOURCES;
        if (!cSession->cTTSChannel->InitializeSynthesis(ttsText))
                return MSPF_CANTTTS;

        ret = DoPlay(ct, nullfile, true, true);
        if (ret != MSPF_OK)
                StopTTS();
        return ret;
}
bool
MediaChannel::HandleEndCompositePrompt(SessionNode *pSession, EndCompositePromptMessage *pMsg, void
*userdata)
{
        if (!VerifySession(pSession))
                return false;
        unsigned long ret = MSPF_OK;

        if (GetState() == S_Idle)
        {
#ifdef NOTIFY_ALL
       cSession->NotifyCompositePromptEnded(pMsg->GetCallToken());
#endif
                return true;
        }
        else {
       error("HandleEndCompositePrompt: MediaChannel is not idle %d", GetState());
                ret = MSPF_NOTIDLE;
    }
        pSession->NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), ret);
        return false;
}
MediaChannel::PromptSegmentStatus
```

```
MediaChannel::PlaySegmentRun(bool First, CallToken& ct)
{
        SessionNode *pSession = cSession;
        Synchronized sync(pSession);
        if (First)
        {
                if (cStoppingCompositePlay)
                {
                        debug("cStoppingCompositePlay is still true for the first segment, wait until it is set to
false");
                        long timeLeft = 5000;
                        long targetTime = timeLeft + ::GetTickCount();
                        while (!pSession->IsGoingAway() &&
                                cStoppingCompositePlay && timeLeft > 0) {
                                sync.wait(timeLeft);
                                timeLeft = targetTime - ::GetTickCount();
                        }
                }
                if (pSession->IsGoingAway())
                {
                        cStoppingCompositePlay = false;
                        cCompositePlayStopped = true;
                        debug("PlaySegmentRun, session is ging away, set cStoppingCompositePlay to false and
cCompositePlayStopped to true");
                        return ePSS_AllSegmentsDone;
                }
                else if (cStoppingCompositePlay)
                {
                        error("previous play is not stopped somehow, cStoppingCompositePlay has not been set to
false");
                        return ePSS_Failure;
                }
                //if (!cCompositePlayStopped)
                //      debug("cCompositePlayStopped is false for the first segment somehow");
                //cStoppingCompositePlay = false;
                //cCompositePlayStopped = true;
                debug("PlaySegmentRun, first segment, set cStoppingCompositePlay to false and
cCompositePlayStopped to true");
        }
        PromptSegment *pSeg = First ? cPromptSegmentMgr.GetFirst() :
        cPromptSegmentMgr.GetNext();
        if (!pSeg && cStoppingCompositePlay)
        {
                cStoppingCompositePlay = false;
                cCompositePlayStopped = true;
                sync.notifyAll();
                debug("PlaySegmentRun, no more segment to play, set cStoppingCompositePlay to false and
cCompositePlayStopped to true");
        }
        if (pSeg)
        {
                if (cStoppingCompositePlay)
                {
                        cStoppingCompositePlay = false;
                        cCompositePlayStopped = true;
                        sync.notifyAll();
```

```
                        debug("PlaySegmentRun, play is stopping, set cStoppingCompositePlay to false and
cCompositePlayStopped to true");
                        return ePSS_AllSegmentsDone;
                }
#ifdef NOTIFY_ALL
                if (First)
            cSession->NotifyCompositePromptPlaying(ct);
#endif
                debug("PlaySegmentRun: before Play");
                if (!pSeg->Play(this, ct))
                {
        // CLEAN UP ANYWAY. M.O.
                cPromptSegmentMgr.Clear();
        SetState(S_Idle);
                        cSession->NotifyMSPFailure(cCurrentCallToken, cMsgSerialNumber,
MSPF_CANTPLAYPROMPTSEGMENT);
                        // redundant GetMSP()->NotifyFailure(cClientID, cCurrentCallToken, cMsgSerialNumber,
MSPF_COMPOSITEPROMPTFAILURE);
                        return ePSS_Failure;
                }
#ifdef NOTIFY_ALL
                cSession->NotifyCompositePromptSegmentPlaying(ct);
#endif
                return ePSS_Playing;
        } else {
                debug("PlaySegmentRun: No Segment");
        }
        return ePSS_AllSegmentsDone;
}
// virtual
bool
MediaChannel::HandlePlayCompositePrompt(SessionNode *pSession, PlayCompositePromptMessage *pMsg, void
*userdata)
{
        if (!VerifySession(pSession))
                return false;
        unsigned long ret = MSPF_OK;

        if (pMsg)
        {
                cMsgSerialNumber = pMsg->GetSerialNo();
                if (GetState() == S_Idle)
                {
        PromptSegmentStatus segmentRunStat = PlaySegmentRun(true, pMsg->GetCallToken());
                        if (segmentRunStat == ePSS_Playing)
                        {
                                cPlayingComposite = true;
                                if (pSpChanTBNotified)
                                        pSpChanTBNotified->NotifyPlayStarted();
        } else if (segmentRunStat == ePSS_AllSegmentsDone) {
          pSession->NotifyCompositePromptPlayDone(cCurrentCallToken);
        }
                        // PlaySegmentRun takes care of the error cases that are related to
                        // the segment play - return true
                        return true;
                }
```

```
                else {
            error("HandlePlayCompositePrompt: medial channel is not idle %d", GetState());
                        ret = MSPF_NOTIDLE;
          }
         }
         else
         {
       error("null message?");
                ret = MSPF_COMPOSITEPROMPTFAILURE;
         }
         pSession->NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), ret);
  return false;
}
// virtual
bool
MediaChannel::NotifyCallAboutToBeDropped(SessionNode *pSession, CallToken& callToken)
{
        ChannelState OldState = GetState();
        debug("NotifyCallAboutToBeDropped, state = %ld.", OldState);
        if (OldState == S_Playing || OldState == S_PlayingTTS || OldState == S_Recording)
        {
                if (OldState == S_Playing || OldState == S_PlayingTTS)
                {
                        SetState(S_StopPlayRequested);
                        if (OldState == S_PlayingTTS)
                                StopTTS();
                        StopAndNotify();
                }
                else
                {
                        debug(" calling StopTransmission(), state = %ld.", OldState);
                        if (!StopTransmission())
                        {
                                SetState(S_StopRecordRequested);
                                debug("in NotifyCallAboutToBeDropped, calling StopAndNotify().");
                                StopAndNotify();
                                debug("in NotifyCallAboutToBeDropped, called StopAndNotify().");
                        }
                        else
                        {
                                debug("in NotifyCallAboutToBeDropped, called StopTransmission.");
                        }
                }
                if (cPlayingComposite)
                {
            cSession->NotifyCompositePromptSegmentPlayDone(cCurrentCallToken);
            cSession->NotifyCompositePromptPlayDone(cCurrentCallToken);
            cPlayingComposite = false;
                }
        }
  // CLEAN UP ANYWAY. M.O.
  cPromptSegmentMgr.Clear();
        SetState(S_Idle);
        CleanupRecordings();
        return true;
}
```

```
// virtual
bool
MediaChannel::HandleReleaseRecording(SessionNode *pSession, CallToken& callToken, RECORDINGID
recordingID)
{
        return cRecordingMgr.Remove(recordingID);
}
// virtual
bool
MediaChannel::GetRecordingFilename(RECORDINGID recordingID, gm::string& filename)
{
        return cRecordingMgr.Lookup(recordingID, filename);
}
// virtual
bool
MediaChannel::HandleRemoveRecording(SessionNode *pSession, RemoveRecordingMessage *pMsg, void *userdata)
{
        if (!VerifySession(pSession))
                return false;
        gm::string filename;
        if (cRecordingMgr.Lookup(pMsg->GetRecordingID(), filename))
        {
                cRecordingMgr.Remove(pMsg->GetRecordingID());
                _unlink(filename);
                return true;
        }
        pSession->NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
MSPF_NOSUCHRECORDING);
  return false;
}
void
MediaChannel::NotifyPlayDone(void)
{
        bool CompletelyDone = true;
        bool PlayFailure = false;
  SessionNode *pSession = cSession;
        if (!VerifySession(pSession))
                return;
  Synchronized sync(pSession);
  if (pSession->IsGoingAway())
    return;
        ChannelState state = GetState();
        debug("NotifyPlayDone: got session lock, state = %d", state);
        if (state == S_Playing || state == S_PlayingTTS || state == S_Idle)
        {
#ifdef NOTIFY_ALL
                if (state == S_Playing)
      cSession->NotifyPlayDone(cCurrentCallToken);
                else
      cSession->NotifyTTSDone(cCurrentCallToken);
#endif
                if (cPlayingComposite)
                {
      cSession->NotifyCompositePromptSegmentPlayDone(cCurrentCallToken);
                        debug("NotifyPlayDone: NotifyCompositePromptSegmentPlayDone done");
                        PromptSegmentStatus segmentRunStat = PlaySegmentRun(false, cCurrentCallToken);
```

```
                        debug("NotifyPlayDone: PlaySegmentRun done");
                        if (segmentRunStat == ePSS_Playing)
                        {
                                CompletelyDone = false;
                        }
                        else if (segmentRunStat == ePSS_AllSegmentsDone)
                        {
                                CompletelyDone = true;
                        }
                        else if (segmentRunStat == ePSS_Failure)
                        {
                                CompletelyDone = false;
                                PlayFailure = true;
                        }
                }
                if (CompletelyDone)
                {
                        if (pSpChanTBNotified)
                                pSpChanTBNotified->NotifyPlayDone();
                        if (cPlayingComposite)
        cSession->NotifyCompositePromptPlayDone(cCurrentCallToken);
                }
                // clean up
                if (CompletelyDone || PlayFailure)
                {
                        cPlayingComposite = false;
                        cPlayingFilename.erase();
      // CLEAN UP ANYWAY. M.O.
                        cPromptSegmentMgr.Clear();
                                SetState(S_Idle);
                }
        }
        else
                error("Got PlayDone when not playing?!?");
}
void
MediaChannel::NotifyStopped(void)
{
        ChannelState state = GetState();
        if (state == S_StopPlayRequested || state == S_StopRecordRequested)
        {
                if (state == S_StopPlayRequested)
                {
#ifdef NOTIFY_ALL
            cSession->NotifyStoppedPlaying(cCurrentCallToken);
#endif
                        if (cPlayingComposite)
                        {
#ifdef NOTIFY_ALL
                // This should not say DONE because
                // we don't know how far it was played.
          cSession->NotifyCompositePromptSegmentPlayDone(cCurrentCallToken);
#endif
          cSession->NotifyCompositePromptPlayDone(cCurrentCallToken);
          //cPlayingComposite = false;
                        }
```

```
                }
                else if (state == S_StopRecordRequested)
                {
            cSession->NotifyStoppedRecording(cCurrentCallToken);
                }
                if (pSpChanTBNotified)
                        pSpChanTBNotified->NotifyPlayStopped();
                if (!cpWriteStreamingPlayFile) // don't erase if it's still used
                        cPlayingFilename.erase();
    // CLEAN UP ANYWAY. M.O.
        cPromptSegmentMgr.Clear();
                SetState(S_Idle);
        }
        NotifySettled();
}
void
MediaChannel::NotifyDTMF(char dtmf)
{
        if (pSpChanTBNotified)
                pSpChanTBNotified->NotifyDTMF(dtmf);
        else {
                unsigned long timestamp = ::GetTickCount();
                SessionNode *pSession = cSession;
                if (!VerifySession(pSession))
                        return;
                Synchronized sync(pSession);
                if (pSession->IsGoingAway())
                        return;
                pSession->NotifyDTMF(cCurrentCallToken, dtmf, timestamp);
        }
}
bool
MediaChannel::StartTransmission(void)
{
        unsigned long ret = MSPF_OK;
        ChannelState state = GetState();
        if (state == S_Idle) {
                if (ret == MSPF_OK) {
                        gm::string filename("dummyfilename");
                        ret = Record(filename, true, -1, -1, -1);
                }
        }
        else {
                ret = MSPF_NOTIDLE;
                error("StartTransmission: MediaChannel not idle %d.",
            GetState());
        }
        if (ret == MSPF_OK)     {
                SetState(S_Recording);
                debug("recording and transmission started.");
                return true;
        }
        return false;
}
bool
MediaChannel::StopTransmission(void)
```

```
{
        unsigned long ret;
        ChannelState state = GetState();
        if (state == S_Recording) {
                SetState(S_StopRecordRequested);
                debug("calling Stop(), state S_StopRecordRequested.");
                ret = Stop();
                diag("called Stop(), ret = %ld.", ret);
                if (ret == MSPF_OK) {
       // CLEAN UP ANYWAY. M.O.
                cPromptSegmentMgr.Clear();
                        SetState(S_Idle);
                        return true;
                }
        }
        else {
                ret = MSPF_NOTHINGTOSTOP;
                diag("cannot stop transmission, state is NOT S_Recording.");
        }
        return false;
}
bool
MediaChannel::RequestNotification(SpeechChannel *pSpeechChannel)
{
        pSpChanTBNotified = pSpeechChannel;
        return true;
}
void
MediaChannel::NotifySample(const unsigned char *pSample, unsigned int numBytes)
{
        // This function is called by the sample transmitter
        if (::WaitForSingleObject(cStreamingWriteMutex, INFINITE) != WAIT_TIMEOUT) {
                // write to file
                if (!cpWriteStreamingPlayFile) {
                        error("Streaming file does not exist.");
                }
                else {
                        if ((fwrite(pSample, sizeof(char), numBytes, cpWriteStreamingPlayFile))
                                < numBytes) {
                                error("Streaming file write failed.");
                        }
                        else {
                                fflush(cpWriteStreamingPlayFile);
                                debug("Wrote %d bytes.", numBytes);
                        }
                }
                ::ReleaseMutex(cStreamingWriteMutex);
                return;
        }
        else {
                return;
        }
}
void
MediaChannel::NotifyEndOfData(int reason)
{
```

```
        debug("NotifyEndOfData:  End of Data Notified.");
        ::SetEvent(cEndOfStreamPlayDataEvent);
        debug("NotifyEndOfData:  Set cEndOfStreamPlayDataEvent.");
}
// virtual
void
MediaChannel::NotifyPropertySet(Property& prop)
{
}
void
MediaChannel::StopTTS(void)
{
  SessionNode *pSession = cSession;
        if (!VerifySession(pSession))
                return;

  debug("in StopTTS(), getting a session lock");
  Synchronized sync(pSession);
  debug("in StopTTS(), got a session lock");
  if (pSession->IsGoingAway())
    return;
        if (cSession->cTTSChannel == NULL)
        {
                error("Can't stop TTS - session has gone away or TTSChannel does not exist.");
                return;
        }
        // do not check for session->IsGoingAway() at
        // this level - it may not be called from client
  debug("in StopTTS(), calling TTSChannel::Release()");
        cSession->cTTSChannel->Release();
  debug("in StopTTS(), called TTSChannel::Release()");
        ::ResetEvent(cEndOfStreamPlayDataEvent);
}
unsigned long
MediaChannel::StopAndNotify(void)
{
        unsigned long ret = Stop();
        if (ret == MSPF_OK) {
                debug("in StopAndNotify, calling NotifyStopped.");
                NotifyStopped();
                debug("in StopAndNotify, called NotifyStopped.");
        }
        return ret;
}
// virtual
void
MediaChannel::NotifyBoundToSession(SessionNode *pSessionNode)
{
        SCChannel::NotifyBoundToSession(pSessionNode);
        if (!pSessionNode && (GetState() == S_Idle))
        {
                debug("in NotifyBoundToSession, calling NotifySettled.");
                NotifySettled();
                debug("in NotifyBoundToSession, called NotifySettled.");
        }
}
```

```
void
MediaChannel::NotifyFailure(void)
{
   // CLEAN UP ANYWAY. M.O.
   cPromptSegmentMgr.Clear();
   SetState(MediaChannel::S_Idle);
   diag("in NotifyFailure() - set state to idle.");
}
```

Document ID: 1.3.6.1.4.1.9.9.24.1.1.2.4

```
/*
 * MediaServiceProvider.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.34 $
 * $Date: 2000/12/07 03:19:48 $
 */
static const char* id="$Id: MediaServiceProvider.cpp,v 1.34 2000/12/07 03:19:48 Mitsuru Oshima Exp $";
#include "MediaServiceProvider.h"
#include "MediaChannel.h"
#include "TTSChannel.h"
#include "MTSServer.h"
#include "TelephonyServiceProvider.h"
#include "VPPMsg/Property.h"
#include "VPPMsg/PlayMsg.h"
#include "VPPMsg/StopPlayMsg.h"
#include "VPPMsg/RecordMsg.h"
#include "VPPMsg/StopRecordMsg.h"
#include "VPPMsg/RemoveRecordingMsg.h"
#include "VPPMsg/TTSMsg.h"
#include "VPPMsg/CreateCompositePromptMsg.h"
#include "VPPMsg/AppendCompositePromptPromptMsg.h"
#include "VPPMsg/AppendCompositePromptSilenceMsg.h"
#include "VPPMsg/AppendCompositePromptTTSMsg.h"
#include "VPPMsg/EndCompositePromptMsg.h"
#include "VPPMsg/PlayCompositePromptMsg.h"
#include "VPPMsg/HandOffRecordingMsg.h"
#include "VPPMsg/PlayingNtfy.h"
#include "VPPMsg/PlayDoneNtfy.h"
#include "VPPMsg/StoppedPlayingNtfy.h"
#include "VPPMsg/RecordingNtfy.h"
#include "VPPMsg/StoppedRecordingNtfy.h"
#include "VPPMsg/TTSPlayingNtfy.h"
#include "VPPMsg/TTSDoneNtfy.h"
#include "VPPMsg/CompositePromptCreatedNtfy.h"
#include "VPPMsg/CompositePromptSegmentAppendedNtfy.h"
#include "VPPMsg/CompositePromptEndedNtfy.h"
#include "VPPMsg/CompositePromptPlayingNtfy.h"
#include "VPPMsg/CompositePromptSegmentPlayingNtfy.h"
#include "VPPMsg/CompositePromptSegmentPlayDoneNtfy.h"
#include "VPPMsg/CompositePromptPlayDoneNtfy.h"
#include "VPPMsg/FailureNtfy.h"
#include "VPPMsg/DTMFNtfy.h"
#include "PromptSegment.h"
#include "SessionNode.h"
#include "VPPUtils/Synchronize.h"
#include "RecordingHandler.h"
MediaServiceProvider::MediaServiceProvider(MTSServer *pServer, const char *logType)
        : ServiceProvider(pServer, M_NTFY_MSPFAILURE, logType)
{
#ifdef LATER
```

```
  if (RecordingHandlerNames && *RecordingHandlerNames)
  {
    HANDLE hDLL = ::LoadLibrary(*RecordingHandlerNames);

    if (hDLL)
    {
      CREATERECORDINGHANDLERFUNC func =
        (CREATERECORDINGHANDLERFUNC)::GetProcAddress(hDLL,
                                    "CreateRecordingHandler");

      if (func)
      {
        RecordingHandlerNode *pNode = new RecordingHandlerNode;

        pNode->cHdl = hDLL;
        pNode->cRecordingHandler = func();
        pNode->cRecordingHandler->Initialize();
        cRecordingHandlers.push_back(pNode);
      }
      else
        return false;
    }
    else
      return false;

    RecordingHandlerNames++;
  }
#endif
}
// virtual
MediaServiceProvider::~MediaServiceProvider(void)
{
  std::list<RecordingHandlerNode *>::iterator rhiter =
    cRecordingHandlers.begin();
  while (rhiter != cRecordingHandlers.end())
  {
    delete (*rhiter)->cRecordingHandler;
    ::FreeLibrary((*rhiter)->cHdl);
    delete (*rhiter);
    rhiter++;
  }
  Shutdown();
}
bool
MediaServiceProvider::Teardown(void)
{
        return true;
}
bool
MediaServiceProvider::HandOffRecording(const gm::string& ServiceName,
                        const gm::string& filename, void *pData, unsigned long DataLength)
{
  std::list<RecordingHandlerNode *>::iterator iter =
    cRecordingHandlers.begin();
  while ((iter != cRecordingHandlers.end()) &&
      (*iter)->cRecordingHandler->GetServiceName() !=  ServiceName)
```

```
    iter++;
  if (iter != cRecordingHandlers.end())
    return (*iter)->cRecordingHandler->HandOff(filename, pData, DataLength);
  return false;
}
```

Document ID: 20020204.0

```
/*
 * Monitor.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.12 $
 * $Date: 2000/12/07 03:22:18 $
 */
static const char* id="$Id: Monitor.cpp,v 1.12 2000/12/07 03:22:18 Mitsuru Oshima Exp $";
#include "Monitor.h"
#include "UtilsLog.h"
#define SIZEOF(x)          (sizeof(x)/sizeof(x[0]))
Monitor::Monitor(void) : owner(NULL), top(0), tail(0), poolsize(0), nest(0), dead(false)
  // ,notifying(0)
{
  mutex = ::CreateMutex(NULL, FALSE, NULL);
  for (int i=0; i<SIZEOF(waiting); i++)  {
    waiting[i] = NULL;
  }
  for (i=0; i<SIZEOF(pool); i++)  {
    pool[i] = NULL;
  }
  // log("monitor done");
}
Monitor::~Monitor(void)
{
  // We may need more cleaner way to destroy the monitor;
  enter();
  // UtilsLog::warning("closing monitor this=%ld, thread id=%ld, top,tail=%d,%d, nest=%d", this, thisThread(), top,
tail, nest);
  notifyAll();
  dead = true;
  ::ReleaseMutex(mutex); // yield(); and join();
  ::WaitForSingleObject(mutex, INFINITE);
/*
  for (int i=0; i<SIZEOF(waiting); i++) {
    if (waiting[i] != NULL) {
      UtilsLog::warning("closing waiting event which has not been cleared %ld, threadid=%ld",
                waiting[i], thisThread());
      ::CloseHandle(waiting[i]);
      waiting[i] = NULL;
    }
  }
*/
  for (int i=0; i<SIZEOF(pool); i++) {
    if (pool[i] != NULL) {
      ::CloseHandle(pool[i]);
      pool[i] = NULL;
    }
  }
  ::ReleaseMutex(mutex); // yield(); and join();
  ::CloseHandle(mutex);
```

```
}
inline
unsigned long
Monitor::thisThread() {
  return ::GetCurrentThreadId();
}
/*
inline
bool
Monitor::check() {
  if (owner != thisThread()) {
    lobj.error("ERROR");
    return false;
  }
  return true;
}
*/
void
Monitor::enter(void)
{
  // log("Monitor:entering");
  if (owner == thisThread()) {
    nest ++;
    return;
  } else if (dead) {
    UtilsLog::warning("trying to enter on dead monitor : thread id=%ld",
                      thisThread());
    return;
  }
  ::WaitForSingleObject(mutex, INFINITE);
  nest ++;
  // log("Monitor:entered");
  owner = thisThread();
}
void
Monitor::leave(void)
{
  if (owner != thisThread()) {
    UtilsLog::error("Invalid State in Moitor::leave, not owner thread");
    return;
  }
  // log("Monitor:leaving");
  nest --;
  if (nest == 0) {
    owner = NULL;
    ::ReleaseMutex(mutex);
  }
  // log("Monitor:left");
}
void
Monitor::returnToPool(HANDLE notify) {
  if (poolsize == SIZEOF(pool)) {
    UtilsLog::warning("EXCEED POOL CAPACITY in Monitor::returnToPool");
    ::CloseHandle(pool);
    return;
  }
```

```
    pool[poolsize] = notify;
    poolsize ++;
}
HANDLE
Monitor::getFromPool() {
    if (poolsize == 0) {
        return::CreateEvent(NULL, true, false, NULL);
    } else {
        poolsize --;
        HANDLE p = pool[poolsize];
        pool[poolsize] = NULL;
        return p;
    }
}
void
Monitor::queue(HANDLE notify) {
    waiting[top] = notify;
    top ++;
    if (top == SIZEOF(waiting)) {
        top = 0;
    }
    if (top == tail) {
        UtilsLog::error("EXCEED WAIT CAPACITY in Monitor::queue");
    }
}
HANDLE
Monitor::dequeue() {
    if (tail == top) {
        return NULL;
    }
    HANDLE notify = waiting[tail];
    waiting[tail] = NULL;
    tail ++;
    if (tail == SIZEOF(waiting)) {
        tail = 0;
    }
    return notify;
}
static void arraycopy(HANDLE* src, int src_pos,
                HANDLE* to, int to_pos, int length)
{
    src = & (src[src_pos]);
    to = & (to[to_pos]);

    if (src == to) {
        return;
    } else if (src < to) {
        for (int i=length-1; i>=0; i--) {
            to[i] = src[i];
        }
    } else {
        memcpy(to, src, length * sizeof(HANDLE));
    }
};
void
Monitor::removeFromWaiting(HANDLE event) {
```

```
    if (tail == top) {
      return;
    }
    for (int i=0; i<SIZEOF(waiting); i++) {
      if (waiting[i] == event) {
        if (i >= tail) {
          int to = tail +1;
          if (to < SIZEOF(waiting)) {
            arraycopy(waiting, tail,
                    waiting, to,
                    i - tail);
            waiting[tail] = NULL;
            tail = to;
          } else if (to == SIZEOF(waiting)) {
            waiting[tail] = NULL;
            tail = 0;
          } else {
            UtilsLog::error("not valid 1. tail=%d, i=%d, top=%d",
                      tail, i, top);
            return;
          }
        } else if (i>=0) {
          int from = i + 1;

          // this check may not needed.
          if (from < SIZEOF(waiting)) {
            arraycopy(waiting, from,
                    waiting, i,
                    top - i);
          }
          waiting[top] = NULL;
          top --;
          if (top <0) {
            top = SIZEOF(waiting) - 1;
          }
        } else {
          int size;
          if (top > tail) {
            size = top - tail;
          } else {
            size = top + (SIZEOF(waiting) - tail);
          }
          size --;
          UtilsLog::error("not valid position. size = %d, top=%d, tail=%d, i=%d" , size ,
                    top, tail, i);
          return;
        }
/*
        int count = 0;
        for (int i=0; i<SIZEOF(waiting); i++) {
          if (waiting[i] != NULL) {
            count ++;
          }
        }
        if (size != count) {
          UtilsLog::error("not valid. size = %d, count=%d, top=%d, tail=%d, i=%d" , size ,
```

```
                    count, top, tail, i);
            }
*/
            return;
        }
    }
/*
            int size;
            if (top > tail) {
               size = top - tail;
            } else {
               size = top + (SIZEOF(waiting) - tail);
            }
            size --;
            int k = i;
            // UtilsLog::info("removing at %d", i);
            if (tail <= i) {
               if (i > top) {
                  UtilsLog::error("not valid 1. tail=%d, i=%d, top=%d",
                           tail, i, top);
               }
               i--;
               for (; tail <= i; i--) {
                  waiting[i+1] = waiting[i];
               }
               waiting[tail] = NULL;
               tail ++;
            } else {
               if (i >= top) {
                  UtilsLog::error("not valid 2. tail=%d, i=%d, top=%d",
                           tail, i, top);
               }
               memcpy(&waiting[i],
                    &waiting[i+1],
                    top - i - 1);
               if (top == 0) {
                  top = SIZEOF(waiting)-1;
               } else {
                  top --;
               }
               waiting[top] = NULL;
            }
            int count = 0;
            for (int i=0; i<SIZEOF(waiting); i++) {
               if (waiting[i] != NULL) {
                  count ++;
               }
            }
            if (size != count) {
               UtilsLog::error("not valid. size = %d, count=%d, top=%d, tail=%d, i=%d" , size ,
                           count, top, tail, k);
            }
            return;
        }
    }
*/
```

```
    // UtilsLog::info("notified %p", event);
}
int
Monitor::yield() {
    if (owner != thisThread()) {
        UtilsLog::error("Invalid State in Monitor::yield, not owner thread");
        return -1;
    }
    owner = NULL;
    int my_nest = nest;
    nest = 0;
    ::ReleaseMutex(mutex);
    return my_nest;
}
void
Monitor::join(int n) {
    ::WaitForSingleObject(mutex, INFINITE);
    if (owner != NULL) {
        UtilsLog::error("Invalid State in Moitor::join(%d), not owner thread",
                nest);
    }
    if (n < 0) {
        UtilsLog::error("Invalid State in Moitor::join(%d)", nest);
    }
    owner = thisThread();
    nest = n;
}
void
Monitor::wait(long duration) {
    if (owner != thisThread()) {
        UtilsLog::error("Invalid State in Monitor::wait : thread id=%ld",
                thisThread());
        return;
    } else if (dead) {
        UtilsLog::warning("trying to wait on dead monitor : thread id=%ld",
                thisThread());
        return;
    }
    HANDLE notify = getFromPool();
    queue(notify);
    owner = NULL;
    int my_nest = nest;
    nest = 0;
    ::ReleaseMutex(mutex);
    ::WaitForSingleObject(notify, duration == 0 ? INFINITE : duration);
/*
    if (dead == false) {
        return;
    }
*/
    ::WaitForSingleObject(mutex, INFINITE);
    ::ResetEvent(notify);
    removeFromWaiting(notify);
    returnToPool(notify);
    if (owner != NULL) {
        UtilsLog::error("Invalid State in Monitor::wait (2)");
```

```
    }
    nest = my_nest;
    owner = thisThread();
}
void
Monitor::notify() {
    if (owner != thisThread()) {
        UtilsLog::error("Invalid State in Monitor::notify");
        return;
    } else if (dead) {
        UtilsLog::error("trying to notify on dead monitor : thread id=%ld",
                thisThread());
    }
    HANDLE event = dequeue();
    if (event != NULL) {
        ::SetEvent(event);
    } else {
        // log("Notify: no one is waiting in ");
    }
}
void
Monitor::notifyAll() {
    if (owner != thisThread()) {
        UtilsLog::error("Invalid State in Monitor::notifyAll");
        return;
    } else if (dead) {
        UtilsLog::error("trying to notify all on dead monitor : thread id=%ld",
                thisThread());
    }
    HANDLE event = NULL;
    while ( (event = dequeue()) != NULL) {
        ::SetEvent(event);
    }
}
```

```
/*
 * SilenceSegment.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:19:53 $
 */
static const char* id="$Id: SilenceSegment.cpp,v 1.5 2000/12/07 03:19:53 Mitsuru Oshima Exp $";
#include "SilenceSegment.h"
#include "MediaChannel.h"
// virtual
SilenceSegment::~SilenceSegment(void)
{
}
// virtual
bool
SilenceSegment::Play(MediaChannel *pMediaChannel,
                     CallToken& ct)
{
        //::Sleep(cDuration);
        return (bool)(pMediaChannel->DoPlaySilence(ct, cDuration) == MSPF_OK);
}
```

```
/*
 * MTSMonitorImpl.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction controlled under your contract with General Magic
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:19:47 $
 *
 * $Id: MTSMonitorImpl.cpp,v 1.2 2000/12/07 03:19:47 Mitsuru Oshima Exp $
 */
static const char* id="$Id: MTSMonitorImpl.cpp,v 1.2 2000/12/07 03:19:47 Mitsuru Oshima Exp $";
#include "MTSMonitorImpl.h"
#include <VPPUtils/Thread.h>
#include <VPPUtils/Synchronized.h>
#include <MTSLib/TelephonyServiceProvider.h>
#include <MTSLib/MTSServer.h>
#include <VPPUtils/ConfigMgr.h>
////////////////////////////////////////////////////////////////////////
DWORD MTSMonitorImpl::m_start_time = GetCurrentTime();
MTSMonitorImpl::MTSMonitorImpl(MTSServer* pMts) :
 Thread(THREAD_PRIORITY_LOWEST,false), POA_mtsmon::MTSMonitor()
{
 _pMTSServer  = pMts;
 _cAgent = NULL;
 _threadId = 0;
 start(); // start notifications processing thread
}
////////////////////////////////////////////////////////////////////////
MTSMonitorImpl::~MTSMonitorImpl()
{
 if(_cAgent) _cAgent->_release();
}
////////////////////////////////////////////////////////////////////////
int
MTSMonitorImpl::activate( PortableServer::POA_var rootPOA, HANDLE hExitEvent )
{
 int result = 0; // OK

 // save MTS ExitEvent
 _hMtsExitEvent = hExitEvent;
 try
 {

 CORBA::PolicyList policies;
 policies.length(1);
 policies[(CORBA::ULong)0] = rootPOA->create_lifespan_policy(PortableServer::PERSISTENT);
 char machine[MAX_COMPUTERNAME_LENGTH];
 DWORD size = MAX_COMPUTERNAME_LENGTH + 1;
 if(!::GetComputerName(machine, &size))
        strcpy(machine, "localhost");
 gm::string inst="0";
 ConfigMgr::GetString("MTS.Monitor","Instance",inst);
 char buff[128];
```

```
 sprintf(buff, "MTSMonitor_%s.%s_poa", machine, inst.c_str());

 PortableServer::POA_var myMonitorPOA = rootPOA->create_POA(buff, rootPOA->the_POAManager(), policies);
 sprintf(buff, "MTSMonitor.%s.%s", machine, inst.c_str());
 cerr << "activating " << buff << " ...";
 // Choose the ID for the servant
 PortableServer::ObjectId_var id = PortableServer::string_to_ObjectId(buff);
 // Activate the servant with the ID on rootPOA
 myMonitorPOA->activate_object_with_id(id, this);
 cerr << "OK" << endl;
 }
 catch(const CORBA::Exception& e)
 {
  cerr << e << endl;
  result = -1;
 }
 return result;
} // activate
////////////////////////////////////////////////////////////////////////////
void
MTSMonitorImpl::run()
{
 MSG msg;
 _threadId = GetCurrentThreadId();
 while(GetMessage(&msg, NULL, WM_USER, WM_USER + 0x7FFF) > 0)
 {
  mtsmon::MTSMonitorMessage data;
  data.wParam = msg.wParam;
  data.lParam = msg.lParam;
  data.message = msg.message;
  data.time = msg.time;
  try
  {
   if(_cAgent) _cAgent->handleNotification(data);
  }
  catch(const CORBA::Exception& e)
  {
   cerr << e << endl;
   _cAgent = 0;
  }
 } // while
} // run
////////////////////////////////////////////////////////////////////////////
void
MTSMonitorImpl::ping()
{
 // nothing to do
}
////////////////////////////////////////////////////////////////////////////
CORBA::Long
MTSMonitorImpl::registerAgent ( mtsmon::MTSMonitorAgent_ptr agent )
{
 CORBA::Long result = 0;
 try
 {
```

```
  if(_cAgent) _cAgent->_release();
  _cAgent = mtsmon::MTSMonitorAgent::_duplicate(agent);
 }
 catch( const CORBA::Exception &ex )
 {
  cerr << "MTSMonitorImpl::registerAgent" << ex << endl;
  result = -1;
 }
 return result;
} // registerAgent
//////////////////////////////////////////////////////////////////////////
CORBA::Long
MTSMonitorImpl::channelControl ( CORBA::Long _ch, CORBA::Long _cmd )
{
 try
 {
        // set on/off channel
 }
 catch( const CORBA::Exception &ex )
 {
  cerr << "MTSMonitorImpl::registerAgent" << ex << endl;
  return -1;
 }
 return 0;
} // channelControl
//////////////////////////////////////////////////////////////////////////
CORBA::Long
MTSMonitorImpl::serviceControl ( CORBA::Long cmd )
{
 long retsult = 0;
 try
 {
  switch(cmd)
  {
        case 0: // start Shutdown MTS
        ::SetEvent(_hMtsExitEvent);
                break;
        case 1: // warm restart TelephonyServiceProvider

                retsult = !server()->GetTelephonyServiceProvider()->Restart();
                break;

        default:
                retsult = -1;
                break;
  }
 }
 catch( const CORBA::Exception &ex )
 {
        cerr << "MTSMonitorImpl::registerAgent" << ex << endl;
        return -1;
 }
 return retsult;
} // serviceControl
//////////////////////////////////////////////////////////////////////////
TelephonyChannel*
```

```
MTSMonitorImpl::getTelephonyChannel( long ch )
{
 TelephonyChannel* pCh = NULL;
 TelephonyServiceProvider* tsp = server()->GetTelephonyServiceProvider();
 std::list<Channel *>::iterator iter = tsp->GetChannels().begin();
 while (iter != tsp->GetChannels().end())
 {
  TelephonyChannel* tmp = ((TelephonyChannel*)(*iter));
  if( tmp->GetID() == ch )
  {
   pCh = tmp;
   break;
  }
  iter++;
 } // while
 return pCh;
}
////////////////////////////////////////////////////////////////////////////
CORBA::Long
MTSMonitorImpl::getServerData(mtsmon::MTSMonitorServerData& _data)
{
 long result = 0;
 try
 {
  _data.uptime = GetCurrentTime() - m_start_time;
  _data.total_channels = 0;
  _data.busy_channels = 0;
  TelephonyServiceProvider* tsp = server()->GetTelephonyServiceProvider();
  std::list<Channel *>::iterator iter = tsp->GetChannels().begin();
  while (iter != tsp->GetChannels().end())
  {
   TelephonyChannel* tmp = ((TelephonyChannel*)(*iter));
   if( tmp->GetState() != TelephonyChannel::S_Disconnected ) _data.busy_channels++;
   _data.total_channels++;
   iter++;
  } // while
  result = 1;
 }
 catch( const CORBA::Exception &ex )
 {
   cerr << "MTSMonitorImpl::registerAgent" << ex << endl;
   return -1;
 }
 return result;
} // getServerData
////////////////////////////////////////////////////////////////////////////
BOOL
MTSMonitorImpl::postMessage( UINT Msg, WPARAM wParam, LPARAM lParam )
{
 if(!_threadId) return FALSE;
 return ::PostThreadMessage(_threadId, Msg, wParam, lParam);
} // postMessage
////////////////////////////////////////////////////////////////////////////
CORBA::Long
MTSMonitorImpl::getChannelData(CORBA::Long _ch, mtsmon::MTSMonitorChannelData& _data)
{
```

```
long result = 0;
try
{
 TelephonyChannel* pCh = getTelephonyChannel(_ch);
 if(pCh)
 {
  _data.state = pCh->GetState();
  _data.status = pCh->GetStatus();

  SessionNode* psn = pCh->getSessionNode();
  if(psn)
  {
        _data.dest_number = psn->cTarget.GetString();
        _data.orig_number = psn->cCallingNumber.GetString();
  }
  else
  {
        _data.dest_number = "";
        _data.orig_number = "";
  }
  result = 1;
 }
}
catch( const CORBA::Exception &ex )
{
  cerr << "MTSMonitorImpl::registerAgent" << ex << endl;
  result = -1;
}
return result;
}
```

```
/*
 * MTSServer.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.30 $
 * $Date: 2000/12/07 03:19:47 $
 */
static const char* id="$Id: MTSServer.cpp,v 1.30 2000/12/07 03:19:47 Mitsuru Oshima Exp $";
#include <windows.h>
#include <stdio.h>
#include "MTSServer.h"
#include "MTSDefs.h"
#include "PromptMgr.h"
#include "TelephonyServiceProvider.h"
#include "MediaServiceProvider.h"
#include "TTSServiceProvider.h"
#include "SpeechServiceProvider.h"
#include "VPPUtils/Synchronize.h"
#include "VPPUtils/ConfigMgr.h"
#include "VPPMsg/Property.h"
#include "SessionNode.h"
#include "VUISessionImpl.h"
#include "Channel.h"
#include "ConnectionMgr.h"
#include "MTSLib/MTSMonitorImpl.h"
MTSLibLog MTSLibLog::log;
class MTSServerImpl;
MTSServer::MTSServer(const gm::string& PromptRoot, MTSServerImpl* impl) : cSessionMgr(this, impl)
{
        MTSLibLog::info("MTSServer created serving the following groups:");
  SessionMgr::SetSessionMgr(&cSessionMgr);
  PromptManager::SetPromptRoot(PromptRoot);
  // DFB_TBD
  //
  // Register with the service finder like "Hi, I'm an MTS and I'm serving"
  // the following groups: GroupName, PhysicalMachineName, Port...
  //

  // cMTS = mts;
  cTSP = 0;
  cMSP = 0;
  cTTSSP = 0;
  cSSP = 0;
  // cNextNodeID = (NODEID)1;
  cSystemProperties.SetProperty("System.ReleaseString", "MTS Release 0.1");
  // MTSMonitor init
  cMonitor = new MTSMonitorImpl(this);
}
// virtual
MTSServer::~MTSServer(void)
{
```

```
    if (cTSP)
    {
      cTSP->CleanupAllResources();
      cTSP->Teardown();
    }
    if (cMSP)
      cMSP->Teardown();
    if (cTTSSP)
      cTTSSP->Teardown();
    if (cSSP)
      cSSP->Teardown();
    delete cTSP;
    delete cMSP;
    delete cTTSSP;
    delete cSSP;
    delete cMonitor;
    if (chTSPDLL)
      ::FreeLibrary(chTSPDLL);
    if (chMSPDLL)
      ::FreeLibrary(chMSPDLL);
    if (chTTSSPDLL)
      ::FreeLibrary(chTTSSPDLL);
    if (chSSPDLL)
      ::FreeLibrary(chSSPDLL);
}
bool
MTSServer::Initialize()
{
    gm::string tspname, tsplibname;
    gm::string mspname, msplibname;
    gm::string sspname, ssplibname;
    gm::string ttsspname, ttssplibname;
    gm::string parent(MTS_REGISTRY_BASE);
    gm::string newparent;
    if (!ConfigMgr::GetString(parent, "Telephony Service Provider", tspname))
      return false;
    newparent = parent;
    newparent += ".Telephony Service Providers.";
    newparent += tspname;
    if (!ConfigMgr::GetString(newparent, "Library", tsplibname))
      return false;
    cTSP = 0;
    cMSP = 0;
    cTTSSP = 0;
    cSSP = 0;
    chTSPDLL = ::LoadLibrary(tsplibname);
    if (chTSPDLL)
    {
      CREATETSPFUNC func =
        (CREATETSPFUNC)::GetProcAddress(chTSPDLL,
                        "CreateTelephonyServiceProvider");

      if (func)
      {
        cTSP = func(this);
        if (!cTSP->Setup() || !cTSP->PostSetup(newparent))
```

```
                        {
                                MTSLibLog::error("Unable to set up %s TSP dll", tsplibname.c_str());
                                printf("Unable to set up %s TSP dll\n", tsplibname.c_str());
        return false;
                        }
    }
    else
                {
                        MTSLibLog::error("Unable to find function CreateTelephonyServiceProvider in %s TSP
dll", tsplibname.c_str());
                        printf("Unable to find function CreateTelephonyServiceProvider in %s TSP dll\n",
tsplibname.c_str());
        return false;
                }

  }
  else
        {
                MTSLibLog::error("Unable to load %s TSP dll", tsplibname.c_str());
                printf("Unable to load %s TSP dll\n", tsplibname.c_str());
                return false;
        }
  if (!ConfigMgr::GetString(parent, "Media Service Provider", mspname))
    return false;
  newparent = parent;
  newparent += ".Media Service Providers.";
  newparent += mspname;
  if (!ConfigMgr::GetString(newparent, "Library", msplibname))
    return false;
  chMSPDLL = ::LoadLibrary(msplibname);
  if (chMSPDLL)
  {
    CREATEMSPFUNC func =
      (CREATEMSPFUNC)::GetProcAddress(chMSPDLL,
                        "CreateMediaServiceProvider");

    if (func)
    {
      cMSP = func(this);
      if (!cMSP->Setup() || !cMSP->PostSetup(newparent))
                        {
                                MTSLibLog::error("Unable to set up %s MSP dll", msplibname.c_str());
                                printf("Unable to set up %s MSP dll\n", msplibname.c_str());
        return false;
                        }
    }
    else
                {
                        MTSLibLog::error("Unable to find function CreateMediaServiceProvider in %s MSP dll",
msplibname.c_str());
                        printf("Unable to find function CreateMediaServiceProvider in %s MSP dll\n",
msplibname.c_str());
        return false;
                }
  }
  else
```

```
        {
                MTSLibLog::error("Unable to load %s MSP dll", msplibname.c_str());
                printf("Unable to load %s MSP dll\n", msplibname.c_str());
    return false;
        }
  if (!ConfigMgr::GetString(parent, "TTS Service Provider", ttsspname))
    return false;

  newparent = parent;
  newparent += ".TTS Service Providers.";
  newparent += ttsspname;
  if (!ConfigMgr::GetString(newparent, "Library", ttssplibname))
    return false;
  chTTSSPDLL = ::LoadLibrary(ttssplibname);
  if (chTTSSPDLL)
  {
    CREATETTSSPFUNC func =
      (CREATETTSSPFUNC)::GetProcAddress(chTTSSPDLL,
                    "CreateTTSServiceProvider");
    if (func)
    {
      cTTSSP = func(this);
      if (!cTTSSP->Setup() || !cTTSSP->PostSetup(newparent))
                    {
                        MTSLibLog::error("Unable to set up %s TTSSP dll", ttssplibname.c_str());
                        printf("Unable to set up %s TTSSP dll\n", ttssplibname.c_str());
        return false;
                    }
    }
    else
            {
                    MTSLibLog::error("Unable to find function CreateTTSServiceProvider in %s TTSSP dll",
ttssplibname.c_str());
                    printf("Unable to find function CreateTTSServiceProvider in %s TTSSP dll\n",
ttssplibname.c_str());
      return false;
            }
  }
  else
      {
            MTSLibLog::error("Unable to load %s TTSSP dll", ttssplibname.c_str());
            printf("Unable to load %s TTSSP dll\n", ttssplibname.c_str());
    return false;
        }

  if (!ConfigMgr::GetString(parent, "Speech Service Provider", sspname))
    return false;
  newparent = parent;
  newparent += ".Speech Service Providers.";
  newparent += sspname;
  if (!ConfigMgr::GetString(newparent, "Library", ssplibname))
    return false;
  chSSPDLL = ::LoadLibrary(ssplibname);
  if (chSSPDLL)
  {
    CREATESSPFUNC func =
```

```
        (CREATESSPFUNC)::GetProcAddress(chSSPDLL,
                                "CreateSpeechServiceProvider");

    if (func)
    {
      cSSP = func(this);
      if (!cSSP->Setup() || !cSSP->PostSetup(newparent))
                                {
                                        MTSLibLog::error("Unable to set up %s SSP dll", ssplibname.c_str());
                                        printf("Unable to set up %s SSP dll\n", ssplibname.c_str());
        return false;
                                }
    }
    else
                {
                                MTSLibLog::error("Unable to find function CreateSpeechServiceProvider in %s SSP dll",
ssplibname.c_str());
                                printf("Unable to find function CreateSpeechServiceProvider in %s SSP dll\n",
ssplibname.c_str());
        return false;
                }
  }
  else
        {
                MTSLibLog::error("Unable to load %s SSP dll", ssplibname.c_str());
                printf("Unable to load %s SSP dll\n", ssplibname.c_str());
    return false;
        }
        cTSP->BeginService();
  cSessionMgr.Initialize();

  return true;
}
bool
MTSServer::PostMonitorMessage( UINT Msg, WPARAM wParam, LPARAM lParam )
{
  return cMonitor->postMessage(Msg, wParam, lParam);
}
MTSMonitorImpl*
MTSServer::GetMTSMonitor()
{
  return cMonitor;
}
```

```
/*
 * MTSServerImpl.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:19:47 $
 */
static const char* id="$Id: MTSServerImpl.cpp,v 1.3 2000/12/07 03:19:47 Mitsuru Oshima Exp $";
#include "MTSServerImpl.h"
#include "OutboundCallRequest.h"
#include "TelephonyServiceProvider.h"
#include "VUISessionImpl.h"
#include "ConnectionMgr.h"
MTSServerImpl::MTSServerImpl(const gm::string &clientname) :
POA_invui::MTSServer(), cClientGroupName("")
{
  cClientGroupName = clientname.c_str();
  cout << "CLIENT GROUP=" << cClientGroupName << endl;
//  clientId = PortableServer::string_to_ObjectId((const char*)clientname);
}
invui::Call*
MTSServerImpl::placeCall(invui::VUIClient_ptr vc, const char* number,
                         const char* ani, const invui::UUIType& uui, invui::TIME timeout)
{
  // TelephonyServiceProvider* TSP = GetTelephonyServiceProvider();
  OutboundCallRequest* request = OutboundCallRequest::Create();
  ConnectionMgr::addConnectionListener(vc, request);
  CDRInfo cdr; // dummy
  unsigned long ret = cTSP->HandlePlaceCall(number, ani, (unsigned char *)uui.get_buffer(), uui.length(), request,
cdr);
  switch (ret) {
  case TSPF_NOAVAILABLECHANNEL:
    OutboundCallRequest::Destroy(request);
    throw invui::NoMoreChannel();
  case TSPF_OK:
    break;
  default:
    OutboundCallRequest::Destroy(request);
    throw invui::TelephonyError();
  }
  VUISessionImpl* newSession = request->GetSession();
  newSession->SetVUIClient(invui::VUIClient::_duplicate(vc));
  // request will destry itself.
  invui::Call_var call = request->GetCall();
  ConnectionMgr::addConnectionListener(vc, newSession);
  return call._retn();
}
void
MTSServerImpl::ping() {
  // sleep
}
```

```
invui::VUIClient_ptr
MTSServerImpl::GetVUIClient() {
   cout << "GETTING CLIENT : GROUP=" << cClientGroupName << endl;
   // Locate an account manager. Give the full POA name and the servant ID.
   char buf[128];
   sprintf(buf, "VUIClient.%s", (const char*)cClientGroupName);
   PortableServer::ObjectId_var id =
      //rtableServer::ObjectId::_duplicate(clientId);
      PortableServer::string_to_ObjectId(buf);

   sprintf(buf, "/VUIClient_%s_poa", (const char*)cClientGroupName);
/*
   CORBA::Any_var policyValue(new CORBA::Any);
   policyValue <<= Messaging::TRANSPARENT;
   CORBA::PolicyList_var policies = new CORBA::PolicyList(1UL);
   policies->length(1UL);
   policies[0UL] = orb->create_policy(Messaging::REBIND_POLICY_TYPE, policyValue);
*/
   try {
      invui::VUIClient_var client = invui::VUIClient::_bind(buf, id);
//       client->set_policy_overrides(policies, CORBA::SET_OVERRIDE);
      return client._retn();
      // return invui::VUIClient::_bind("/JDE_poa", clientId);
      //       return invui::VUIClient::_duplicate(vc);
   } catch (const CORBA::Exception &ex) {
      // log
      cerr << "MTSServer::GetVUIClient" << ex << endl;
      return NULL;
   }
}
```

```
/*
 * NLInterpretation.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:50 $
 */
static const char* id="$Id: NLInterpretation.cpp,v 1.3 2000/12/07 03:21:50 Mitsuru Oshima Exp $";
#include "NLInterpretation.h"
#include "VPPMsg.h"
NLInterpretation::NLInterpretation(void)
{
}
NLInterpretation::~NLInterpretation(void)
{
        std::list<Slot *>::iterator iter =  cSlots.begin();
        while (iter != cSlots.end())
        {
                Slot *pSlot = *iter;
                delete pSlot;
                iter++;
        }
}
gm::string&
NLInterpretation::GetGrammarKey()
{
        return grammar_key;
}
void
NLInterpretation::SetGrammarKey(gm::string& key) {
        grammar_key = key.c_str();
}
unsigned short
NLInterpretation::GetNumSlots(void)
{
        return (unsigned short)cSlots.size();
}
Slot *
NLInterpretation::GetSlot(int index)
{
        std::list<Slot *>::iterator iter =  cSlots.begin();
        int i = 0;
        while (iter != cSlots.end() && i < index)
        {
                i++;
                iter++;
        }
        if (iter != cSlots.end())
                return *iter;
        return 0;
}
```

```
void
NLInterpretation::AddSlot(Slot *pSlot)
{
	cSlots.push_back(pSlot);
}
void
NLInterpretation::Serialize(VPPMessage *pMsg)
{
	pMsg->InsertUshort(GetNumSlots());
	std::list<Slot *>::iterator iter =  cSlots.begin();
	while (iter != cSlots.end())
	{
		(*iter)->Serialize(pMsg);
		iter++;
	}
}
void
NLInterpretation::Deserialize(VPPMessage *pMsg)
{
	unsigned short num = pMsg->ExtractUshort();
	cSlots.clear();
	for (unsigned short i = 0; i < num; i++)
	{
		Slot *pSlot = new Slot;
		pSlot->Deserialize(pMsg);
		AddSlot(pSlot);
	}
}
```

```
/*
 * NuanceSChannel.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.35 $
 * $Date: 2000/12/07 03:24:27 $
 */
static const char* id="$Id: NuanceSChannel.cpp,v 1.35 2000/12/07 03:24:27 Mitsuru Oshima Exp $";
#include <windows.h>
#include <stdio.h>
#include "VPPMsg/VPPDefs.h"
#include "VPPUtils/ConfigFile.h"
#include "VPPUtils/SecureHash.h"
#include "NuanceSChannel.h"
#include "NuanceSSP.h"
#include "VRSAPIClient.h"
#include <MTSLib/SessionNode.h>
NUAN_SSPLog NUAN_SSPLog::log;
class GrammarSetTable : public Hashtable<char*, GrammarSet*> {
public:
  GrammarSetTable() : Hashtable<char*, GrammarSet*>() {}
  virtual ~GrammarSetTable() {
    clear();
  }
  int _hash(char* value) {
    int h = strlen(value) + value[0];
    return h;
  }
  char* _copy(char* value) {
    char* copy = new char[strlen(value)+1];
    strcpy(copy, value);
    return copy;
  }
  bool _keyEquals(char* k1, char* k2) {
    return strcmp(k1, k2) == 0;
  }
  bool _valueEquals(GrammarSet* v1, GrammarSet* v2) {
    return v1 == v2;
  }
  void _deleteKey(char* key) { // no output
    delete[] key;
  }
};
char*
Hashtable<char*,GrammarSet*>::NULL_KEY = NULL;
GrammarSet*
Hashtable<char*,GrammarSet*>::NULL_VALUE = NULL;
/**
 * This is nuance specific, so this should be moved to NuanceSpeechChannel.
 */
char*
```

```
NuanceSChannel::cDynamicGrammarNames[26] = {
    "Dynamic1",
    "Dynamic2",
    "Dynamic3",
    "Dynamic4",
    "Dynamic5",
    "Dynamic6",
    "Dynamic7",
    "Dynamic8",
    "Dynamic9",
    "Dynamic10",
    "Dynamic11",
    "Dynamic12",
    "Dynamic13",
    "Dynamic14",
    "Dynamic15",
    "Dynamic16",
    "Dynamic17",
    "Dynamic18",
    "Dynamic19",
    "Dynamic20",
    "Dynamic21",
    "Dynamic22",
    "Dynamic23",
    "Dynamic24",
    "Dynamic25",
    "Dynamic26",
};
char*
NuanceSChannel::cDynamicGrammarRefs[26] = {
    "DynaRef1",
    "DynaRef2",
    "DynaRef3",
    "DynaRef4",
    "DynaRef5",
    "DynaRef6",
    "DynaRef7",
    "DynaRef8",
    "DynaRef9",
    "DynaRef10",
    "DynaRef11",
    "DynaRef12",
    "DynaRef13",
    "DynaRef14",
    "DynaRef15",
    "DynaRef16",
    "DynaRef17",
    "DynaRef18",
    "DynaRef19",
    "DynaRef20",
    "DynaRef21",
    "DynaRef22",
    "DynaRef23",
    "DynaRef24",
    "DynaRef25",
    "DynaRef26",
```

```
};
NuanceSChannel::NuanceSChannel(ServiceProvider *pServiceProvider, const char *logType, int id)
   : SpeechChannel(pServiceProvider, logType, id, true), cDynamicGrammarState(DG_NONE),
cNuanceStatus(NUANCE_OK), cTopLevelGrammar(".DynamicTop")
{
   cVRSAPI = 0;
   cState = S_Nonfunctional;
   cGrammarSets = new GrammarSetTable();
}
// virtual
NuanceSChannel::~NuanceSChannel(void)
{
   {
      Synchronized sync(cGrammarSets);

      Hashtable<char*, GrammarSet*>::Iterator itr = cGrammarSets->iterator();
      while (itr.next()) {
         delete itr.value();
      }
   }
   delete cGrammarSets;
   if (cVRSAPI)
      cVRSAPI->Teardown();
   delete cVRSAPI;
}
bool
NuanceSChannel::Setup(unsigned int id, NuanceConfig *config)
{
         NuanceStatus status;
         cVRSAPI = new VRSAPIClient(this);
         AudioFormat af;
         af.rate = 8000;
         af.bits = 8;
         af.encoding = AudioEncodingMulaw;
         if (cVRSAPI->Setup(id, config, &af, status) && status == NUANCE_OK)
         {
                  cState = S_Idle;
         }
         else
         {
                  if (status == NUANCE_LICENSE_CHECK_FAILED)
                           cState = S_Unlicensed;
                  delete cVRSAPI;
                  cVRSAPI = 0;
         }
         return true;
}
// virtual
unsigned long
NuanceSChannel::GetGrammarHandle(DynamicGrammar& gram, GrammarHandle& handle)
{
   switch (gram.GetGrammarType()) {
   case DynamicGrammar::TEXT: {
      TextGrammar* tgram = (TextGrammar*)&gram;
      return GetGrammarHandle(tgram->key, tgram->grammar, handle);
   }
```

```
    case DynamicGrammar::SET:
      return GetGrammarSetHandle((GrammarSet&)gram, handle);
    }
    return SSPF_SYSTEMFAILURE;
}
unsigned long
NuanceSChannel::GetGrammarSetHandle(const GrammarSet& set,
                                    GrammarHandle& handle) {
    gm::string key = "";
    gm::string gsl = "[";
    int dyn = 0;
    char weight_buf[16];
    std::list<GrammarItem *>::const_iterator itr = set.begin();
    while (itr != set.end()) {
      GrammarItem* h = (*itr);
      if (h->handle.cHandleType == GrammarHandle::DYNAMIC) {
        key += h->handle.key.c_str();
        gsl += cDynamicGrammarRefs[dyn++];
        if (h->weight >0) {
          gsl += "~";
          ::_snprintf(weight_buf, 15, "%#.3f", h->weight);
          gsl += weight_buf;
          key += weight_buf;
        }
        gsl += " {<namespace \"";
        gsl += h->handle.key.c_str();
        gsl += "\">} ";
      } else {
        key += h->handle.key.c_str();
        gsl += h->handle.key.c_str();
        if (h->weight >0) {
          gsl += "~";
          ::_snprintf(weight_buf, 15, "%#.3f", h->weight);
          gsl += weight_buf;
          key += weight_buf;
        }
        gsl += " {<namespace \"";
        gsl += h->handle.key.c_str();
        gsl += "\">} ";
      }
      itr ++;
    }
    gsl += "]";
          key = SecureHash::hash(key);
    cGrammarSets->put((char*)key.c_str(), new GrammarSet(set));
    return GetGrammarHandle(key, gsl, handle);
}
unsigned long
NuanceSChannel::GetGrammarHandle(const gm::string& key,
                         const gm::string& gsl,
                         GrammarHandle& handle)
{
    SessionNode* pSession = cSession;
    if (pSession == NULL) {
      return SSPF_SYSTEMFAILURE;
    }
```

```
Synchronized sync(pSession);
if (cDynamicGrammarState != DG_NONE) {
   error("GetGrammarHandle: dynamic grammar state is not idle %d",
       cDynamicGrammarState);
   return SSPF_NOTIDLE;
}
cDynamicGrammarState = DG_QUERY;
unsigned long ret =
   cVRSAPI->QueryDynamicGrammarContents(key,
                              gm::string::Nullref());
while (pSession->IsGoingAway() == false &&
     cDynamicGrammarState == DG_QUERY) {
   sync.wait();
}
if (cDynamicGrammarState == DG_NOTFOUND) {
   cDynamicGrammarState = DG_CREATE;
   info("creating dynamic grammar: key=%s, gsl=%s", (const char*)key, (const char*)gsl);
       ret = cVRSAPI->CreateDynamicGrammar(key, gsl);
   if (ret != SSPF_OK) {
      return ret;
   }
   while (pSession->IsGoingAway() == false &&
       cDynamicGrammarState == DG_CREATE) {
      sync.wait();
   }
}
if (cDynamicGrammarState != DG_SUCCESS) {
   cDynamicGrammarState = DG_NONE;
   return SSPF_CANTRECOGNIZE;
}
cDynamicGrammarState = DG_NONE;
handle.cHandleType = GrammarHandle::DYNAMIC;
handle.key = key.c_str();
return SSPF_OK;
}
// virtual
unsigned long
NuanceSChannel::StartRecognition(const GrammarHandle& handle)
{
   if (cState == S_Idle)
   {
      gm::string grammar_name;
      unsigned long ret = SSPF_OK;
      if (handle.cHandleType == GrammarHandle::DYNAMIC) {
         grammar_name = cTopLevelGrammar.c_str();
         ret = InsertGrammars(handle);
         if (ret != SSPF_OK) {
            return ret;
         }
      } else {
         grammar_name = (const char*)handle.key;
      }
      diag("Calling cVRSAPI->StartRecognition. : %s", grammar_name.c_str());
      ret = cVRSAPI->StartRecognition(grammar_name);
      diag("Called cVRSAPI->StartRecognition.");
      if (ret == SSPF_OK)
```

```
    {
      cState = S_Recognizing;
      info("Recognition started. ");
    }
    else
    {
      error("Cannot start recognition - cVRSAPI->StartRecognition failed, ret = %ld.", ret);
    }
    return ret;
  }
  else if (cState == S_Unlicensed)
  {
    error("Cannot start recognition - state is unlicensed.");
    return SSPF_NOLICENSE;
  }
  else
  {
    error("Cannot start recognition - state is not idle : state=%d", cState);
    return SSPF_CANTRECOGNIZE;
  }
}
// virtual
unsigned long
NuanceSChannel::PushSamples(const unsigned char *pSamples, unsigned int numbytes)
{
        return cVRSAPI->PushSamples(pSamples, numbytes);
}
// virtual
unsigned long
NuanceSChannel::StopRecognition(void)
{
        if (cState == S_Recognizing)
        {
                diag("Calling cVRSAPI->StopRecognition.");
                unsigned long ret = cVRSAPI->StopRecognition();
                diag("Called cVRSAPI->StopRecognition.");
                if (ret == SSPF_OK)
                {
                        info("Recognition stop.");
                }
                else
                {
                        error("cVRSAPI->StopRecognition failed, set to S_WaitingForResults anyway, ret=%ld",
ret);
                }
                cState = S_WaitingForResults;
                return ret;
        }
        else if (cState == S_Unlicensed)
        {
                error("SSPF_NOLICENSE.");
                return SSPF_NOLICENSE;
        }
        else
        {
                warning("Cannot stop recognition because state is not S_Recognizing.");
```

```
                return SSPF_CANTRECOGNIZE;
        }
}
// virtual
unsigned long
NuanceSChannel::AbortRecognition(void)
{
        if (cState == S_Recognizing)
        {
                diag("Calling cVRSAPI->AbortRecognition.");
                unsigned long ret = cVRSAPI->AbortRecognition();
                diag("Called cVRSAPI->AbortRecognition.");
                if (ret == SSPF_OK)
                {
                        info("cVRSAPI->AbortRecognition succeeded - state set to idle.");
                }
                else
                {
                        warning("cVRSAPI->AbortRecognition failed, set to idle anyway, ret=%ld", ret);
                }
                cState = S_Idle;
                return ret;
        }
  else if (cState == S_WaitingForResults)
  {
    cState = S_Idle;
        info("Abort recognition while in S_WaitingForResults - state set to idle.");
    return SSPF_OK;
  }
        else if (cState == S_Unlicensed)
        {
                error("SSPF_NOLICENSE.");
                return SSPF_NOLICENSE;
        }
        else
        {
                warning("Cannot abort recognition because state is not S_Recognizing.");
                return SSPF_CANTRECOGNIZE;
        }
}
void
NuanceSChannel::NotifyDoneRecognizing(RecognitionResults *pResults,
        RecognitionDoneNotification::RecognitionStatus status)
{
        info("Got NotifyDoneRecognizing");
        if (cState == S_WaitingForResults)
        {
                cState = S_Idle;
                diag("In NotifyDoneRecognizing calling NotifyRecognitionDone - state set to idle.");
                NotifyRecognitionDone(pResults, status);
                diag("In NotifyDoneRecognizing called NotifyRecognitionDone - state set to idle.");
        }
}
// virtual
unsigned long
NuanceSChannel::RecognizeString(const GrammarHandle& handle,
```

```
                    const gm::string& str)
{
  if (cState == S_Idle)
  {
    gm::string grammar_name;
    unsigned long ret = SSPF_OK;
    if (handle.cHandleType == GrammarHandle::DYNAMIC) {
      grammar_name = ".DynamicTop";
      ret = InsertGrammars(handle);
      if (ret != SSPF_OK) {
        return ret;
      }
    } else {
      grammar_name = (const char*)handle.key;
    }
    ret = cVRSAPI->RecognizeString(grammar_name, str);
    if (ret == SSPF_OK)
      cState = S_WaitingForResults;
    return ret;
  }
  else if (cState == S_Unlicensed)
    return SSPF_NOLICENSE;
  else
    return SSPF_CANTRECOGNIZE;
}
unsigned long
NuanceSChannel::NuanceToSSPF(NuanceStatus status)
{
  switch(status)
  {
  case NUANCE_OK:
    return SSPF_OK;
  case NUANCE_UNKNOWN_DATABASE_PROVIDER:
  case NUANCE_DATABASE_PROVIDER_NOT_SUPPORTED:
  case NUANCE_CANNOT_OPEN_DATABASE:
  case NUANCE_DATABASE_PROVIDER_NOT_INSTALLED:
  case NUANCE_DB_HANDLE_ALREADY_IN_USE:
    return SSPF_DATABASEERROR;

  case NUANCE_NON_EXISTENT_RECORD:
    return SSPF_UNKNOWNGRAMMAR;

  default:
    return SSPF_SYSTEMFAILURE;
  }
}
unsigned long
NuanceSChannel::InsertGrammars(const GrammarHandle& handle)
{
  diag("Inserting Grammars for %s", handle.key.c_str());
  SessionNode* pSession = cSession;
  if (pSession == NULL) {
    error("Cannot insert grammars : no session");
    return SSPF_SYSTEMFAILURE;
  }
  if (cDynamicGrammarState != DG_NONE) {
```

```
    error("GetGrammarHandle: dynamic grammar state is not idle %d",
        cDynamicGrammarState);
    return SSPF_NOTIDLE;
  }
  if (handle.cHandleType == GrammarHandle::BUILTIN) {
    Property prop("Speech.ActiveGrammarNames", (const char*)handle.key);
    pSession->SetProperty(prop);
    return SSPF_OK;
  }
  cDynamicGrammarState = DG_INSERT;
  cNumberOfInsertedGrammars = 0;
  gm::string active;
  GrammarSet* set = cGrammarSets->get((char*)handle.key.c_str());
  unsigned long ret;
  if (set != NULL) {
    char weight_buf[16];
    active = "[ ";
    if (set->size() == 0) {
      error("InsertGrammars : empty grammar set ");
      return SSPF_UNKNOWNGRAMMAR;
    }
    char* delim="";
    std::list<GrammarItem *>::iterator itr = set->begin();
    while (itr != set->end()) {
      GrammarItem* h = (*itr);
      if (h->handle.cHandleType == GrammarHandle::DYNAMIC) {
        active += (const char*)delim;
        active += "DYNAMIC_GSL(";
        active += (const char*)h->handle.key;
        if (h->weight >0) {
          active += ")";
          active += "*";
          ::_snprintf(weight_buf, 15, "%#.3f", h->weight);
          active += weight_buf;
        } else {
          active += ")";
        }
        const char* name = cDynamicGrammarNames[cNumberOfInsertedGrammars++];
        ret = cVRSAPI->InsertDynamicGrammar(h->handle.key, name);
        if (ret != SSPF_OK) {
          return ret;
        }
      } else {
        active += (const char*)delim;
        active += (const char*)h->handle.key;
        if (h->weight >0) {
          active += "*";
          ::_snprintf(weight_buf, 15, "%#.3f", h->weight);
          active += weight_buf;
        }
      }
      delim = "|";
      itr++;
    }
    active += " ]";
  } else {
```

```
    active = (const char*)handle.key;
  }
  Property prop("Speech.ActiveGrammarNames", (const char*)active);
  pSession->SetProperty(prop);
  cNumberOfInsertedGrammars ++;
  ret = cVRSAPI->InsertDynamicGrammar(handle.key, "DynamicRoot");
  if (ret != SSPF_OK) {
    return ret;
  }
  Synchronized sync(pSession);
  while (pSession->IsGoingAway() ==false &&
      cNumberOfInsertedGrammars &&
      cDynamicGrammarState == DG_INSERT) {
    sync.wait();
  }
  if (cDynamicGrammarState != DG_SUCCESS) {
    cDynamicGrammarState = DG_NONE;
    error("InsertGrammars : could not insert grammars ");
    return NuanceToSSPF(cNuanceStatus);
  }
  cDynamicGrammarState = DG_NONE;
  return SSPF_OK;
}
void
NuanceSChannel::NotifyDynamicGrammarCreated(NuanceStatus status,
                                const gm::string& db_key)
{
  info("Got NotifyDynamicGrammarCreated()");
  SessionNode* pSession = cSession;
  if (pSession == NULL) {
    return;
  }
  Synchronized sync(pSession);
  if (cDynamicGrammarState == DG_CREATE) {
    cNuanceStatus = status;
    if (status == NUANCE_OK) {
      debug("NotifyDynamicGrammarCreated : key=%s",
          (const char*)db_key);
      cDynamicGrammarState = DG_SUCCESS;
    } else {
      error("NotifyDynamicGrammarCreated : key=%s : nuance status=%d",
          (const char*)db_key, status);
      cDynamicGrammarState = DG_ERROR;
    }
    sync.notifyAll();
  } else {
    warning("NotifyDynamicGrammarCreated : key=%s, called in wrong state",
        (const char*)db_key);
  }
}
void
NuanceSChannel::NotifyDynamicGrammarDeleted(NuanceStatus status,
                                const gm::string& db_key)
{
  info("Got NotifyDynamicGrammarDeleted()");
  // nothing for now
```

```
}
void
NuanceSChannel::NotifyPhraseAddedToDynamicGrammar(NuanceStatus status,
                                const gm::string& db_key,
                                const gm::string& phrase_id)
{
  info("Got NotifyPhraseAddedToDynamicGrammar()");
  // nothing for now
}
void
NuanceSChannel::NotifyPhraseRemovedFromDynamicGrammar(NuanceStatus status,
                                  const gm::string& db_key,
                                  const gm::string& phrase_id)
{
  info("Got NotifyPhraseRemovedFromDynamicGrammar()");
  // nothing for now
}
void
NuanceSChannel::NotifyDynamicGrammarContents(NuanceStatus status,
                              const gm::string& db_key,
                              const gm::string& phrase_id,
                              const gm::string& phrase_contents)
{
  SessionNode* pSession = cSession;
  if (pSession == NULL) {
    error("NotifyDynamicGrammarContents : nosession for key=%s",
        (const char*)db_key);
    return;
  }
  Synchronized sync(pSession);
  if (cDynamicGrammarState == DG_QUERY) {
    cNuanceStatus = status;
    cDynamicGrammarState = DG_ERROR;
    switch (status) {
    case NUANCE_OK:
      cDynamicGrammarState = DG_SUCCESS;
      debug("NotifyDynamicGrammarContents : Success, key=%s",
          (const char*)db_key);
      break;
    case NUANCE_UNKNOWN_DATABASE_PROVIDER:
      error("NotifyDynamicGrammarContents : Unknown Database Provider, key=%s", db_key.c_str());
      break;
    case NUANCE_DATABASE_PROVIDER_NOT_SUPPORTED:
      error("NotifyDynamicGrammarContents : Database Provider Not Supported, key=%s", db_key.c_str());
      break;
    case NUANCE_CANNOT_OPEN_DATABASE:
      error("NotifyDynamicGrammarContents : Cannot Open Database, key=%s", db_key.c_str());
      break;
    case NUANCE_DATABASE_PROVIDER_NOT_INSTALLED:
      error("NotifyDynamicGrammarContents : Database Provider Not Installed, key=%s", db_key.c_str());
      break;
    case NUANCE_DB_HANDLE_ALREADY_IN_USE:
      error("NotifyDynamicGrammarContents : DB Handle Already In Use, key=%s", db_key.c_str());
      break;
    case NUANCE_NON_EXISTENT_RECORD:
      cDynamicGrammarState = DG_NOTFOUND;
```

```
        debug("NotifyDynamicGrammarContents : Cannot Find Dynamic Grammar, key=%s",  db_key.c_str());
        break;
      default:
        error("NotifyDynamicGrammarContents : Failure, key=%s",
            (const char*)db_key);
        break;
      }
      sync.notifyAll();
    } else {
      warning("NotifyDynamicGrammarContents : called in wrong state: key = %s",
          (const char*)db_key);
    }
}
void
NuanceSChannel::NotifyDynamicGrammarInserted(NuanceStatus status,
                                const gm::string& db_key,
                                const gm::string& grammar_name)
{
    info("Got NotifyDynamicGrammarInserted()");
    SessionNode* pSession = cSession;
    if (pSession == NULL) {
      error("NotifyDynamicGrammarInserted : nosession for key=%s",
          (const char*)db_key);
      return;
    }
    Synchronized sync(pSession);
    if (cDynamicGrammarState == DG_INSERT) {
      cNuanceStatus = status;
      if (status == NUANCE_OK) {
        cNumberOfInsertedGrammars --;
        diag("NotifyDynamicGrammarInserted : key=%s, name=%s, NumberOfInsertedGrammars = %d",
            (const char*)db_key, (const char*)grammar_name,
             cNumberOfInsertedGrammars);
        if (cNumberOfInsertedGrammars == 0) {
          cDynamicGrammarState = DG_SUCCESS;
          sync.notifyAll();
        }
      } else {
        cDynamicGrammarState = DG_ERROR;
        error("NotifyDynamicGrammarInserted : key=%s, name=%s, error",
            (const char*)db_key, (const char*)grammar_name);
        sync.notifyAll();
      }
    } else {
      warning("NotifyDynamicGrammarInserted : key=%s, name=%s, called in wrong state",
          (const char*)db_key, (const char*)grammar_name);
    }
}
// virtual
void
NuanceSChannel::NotifyBoundToSession(SessionNode *pSessionNode)
{
    cTopLevelGrammar = ".DynamicTop";
    SpeechChannel::NotifyBoundToSession(pSessionNode);
    cVRSAPI->NotifyBoundToSession(pSessionNode);
    if (pSessionNode == NULL) {
```

```
    // deleting dynamic grammar set
    Hashtable<char*,GrammarSet*>::Iterator itr = cGrammarSets->iterator();
    while (itr.next()) {
      delete itr.value();
    }
    debug("Clearing Grammar Set");
    cGrammarSets->clear();
    cDynamicGrammarState = DG_NONE; // cleanup
  }
}
// virtual
bool
NuanceSChannel::StartLogging(void)
{
        return cVRSAPI->StartLogging();
}
// virtual
bool
NuanceSChannel::LogUtteranceFilename(void)
{
        gm::string uttFilename;
        //cVRSAPI->GetUtteranceFilename(uttFilename);
        //gm::string printString;
        //printString += "Utterance Filename: " + uttFilename + ", Recording filename: " + cRecordingFilename;
        if (NuanceSSP::GetLogRecordingFilename())
                cVRSAPI->AttachRecordingFilename(cRecordingFilename);
        //debug("UttFilename: %s.", uttFilename);
        //debug("RecFilename: %s.", cRecordingFilename);
        return cVRSAPI->LogUtteranceFilename();
}
// virtual
bool
NuanceSChannel::StopLogging(void)
{
        return cVRSAPI->StopLogging();
}
// virtual
void
NuanceSChannel::NotifyPropertySet(Property& prop)
{
  if (prop.GetName() == "Nuance.TopLevelGrammar") {
    cTopLevelGrammar = prop.GetValue().c_str();
  } else {
        SpeechChannel::NotifyPropertySet(prop);
        cVRSAPI->NotifyPropertySet(prop);
  }
}
void
NuanceSChannel::NotifySettled(void)
{
  SpeechChannel::NotifySettled();
}
```

```
/*
 * NuanceSSP.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.18 $
 * $Date: 2000/12/07 03:24:27 $
 */
static const char* id="$Id: NuanceSSP.cpp,v 1.18 2000/12/07 03:24:27 Mitsuru Oshima Exp $";
#include "NuanceSSP.h"
#include "NuanceSChannel.h"
#include "MTSLib/MTSServer.h"
#include "MTSLib/TelephonyServiceProvider.h"
#include "VPPUtils/ConfigFile.h"
#include "VPPUtils/ConfigMgr.h"
NuanceSSP *NuanceSSP::cInstance = 0;
gm::string NuanceSSP::cPackagesSpecificationFile("nuance.packages");
gm::string NuanceSSP::cParametersFile("nuance.parameters");
bool NuanceSSP::cLogRecordingFilename = false;
const char const *NuanceSSP::cLogType="NUAN_SSP";
NuanceSSP::NuanceSSP(MTSServer *pServer)
        : SpeechServiceProvider(pServer, cLogType)
{
        GetMTSServer()->GetSystemProperties().SetProperty("Speech.ReleaseString",
                "Nuance VRSAPI SSP Release 0.1");
        gm::string str;

        if (ConfigMgr::GetString(MTS_REGISTRY_BASE ".Speech Service Providers.Nuance",
                "PackagesSpecificationFile", str))
                cPackagesSpecificationFile = str.c_str();
        if (ConfigMgr::GetString(MTS_REGISTRY_BASE ".Speech Service Providers.Nuance",
                "ParametersFile", str))
                cParametersFile = str.c_str();
        gm::string regStr;
        if (ConfigMgr::GetString(MTS_REGISTRY_BASE ".Speech Service Providers.Nuance",
                "LogRecordingFilename", regStr))
        {
                if (!::stricmp(regStr.c_str(), "true"))
                        cLogRecordingFilename = true;
        }
}
// virtual
NuanceSSP::~NuanceSSP(void)
{
        diag("NuanceSSP", "NuanceSSP is shutting down.");
   Shutdown();
        info("NuanceSSP", "NuanceSSP has been shut down.");
}
// virtual
bool
NuanceSSP::Setup(void)
{
```

```
        ConfigFile packages(cPackagesSpecificationFile);
        // Build a command line-like structure that specifies all the desired packages, plus
        // a pointer to the nuance.config configuration file
        //
        char **av = new char *[((packages.GetNumLines() * 2) + 2)];
        int ndx = 0;
        av[ndx++] = "-config";
        av[ndx++] = (char *)(const char *)cParametersFile;
        for (unsigned short s = 0; s < packages.GetNumLines(); s++)
        {
                av[ndx++] = "-package";
                av[ndx++] = (char *)packages.GetLine(s);
        }
        NuanceStatus status;
        NuanceConfig *config = NuanceConfigBuildFromCommandLine(&ndx, av, 0, &status);
        delete [] av;
        if (status != NUANCE_OK)
                return false;
        //
        // For each telephony channel, try to allocate a speech channel
        //
        std::list<Channel *>::size_type numchn = GetMTSServer()->
                GetTelephonyServiceProvider()->GetNumChannels();
        unsigned int id = 1;
        for (int i = 0; i < (int)numchn; i++)
        {
                NuanceSChannel *pNewChannel = new NuanceSChannel(this, cLogType, i);
                pNewChannel->Setup(id++, config);
                AddChannel(pNewChannel);
        }
        return true;
}
// virtual
bool
NuanceSSP::Teardown(void)
{
        // Channels are deleted by superclass
        //
        return true;
}
void
NuanceSSP::SetInstance(NuanceSSP *ptr)
{
        cInstance = ptr;
}
NuanceSSP *
NuanceSSP::GetInstance(void)
{
        return cInstance;
}
SpeechServiceProvider *
CreateSpeechServiceProvider(MTSServer *pServer)
{
        NuanceSSP::SetInstance(new NuanceSSP(pServer));
        return NuanceSSP::GetInstance();
}
```

```
/*
 * OneCallTokenMsgBase.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:21:51 $
 */
static const char* id="$Id: OneCallTokenMsgBase.cpp,v 1.5 2000/12/07 03:21:51 Mitsuru Oshima Exp $";
#include "OneCallTokenMsgBase.h"
// virtual
void
OneCallTokenMessageBase::Serialize(void)
{
        VPPMessage::Serialize();
        cCallToken.Serialize(this);
}
// virtual
void
OneCallTokenMessageBase::Deserialize(void)
{
        VPPMessage::Deserialize();
        cCallToken.Deserialize(this);
}
#if 0
// virtual
VirtualMachineID&
OneCallTokenMessageBase::GetMTSVirtualMachineID(void)
{
        return cCallToken.GetMTSVirtualMachineID();
}
#endif
```

```
/*
 * OutboundCallRequest.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.8 $
 * $Date: 2000/12/07 03:19:49 $
 */
static const char* id="$Id: OutboundCallRequest.cpp,v 1.8 2000/12/07 03:19:49 Mitsuru Oshima Exp $";
#include "OutboundCallRequest.h"
#include "ConnectionMgr.h"
#include "VUISessionImpl.h"
#include "VPPUtils/Synchronized.h"
#include "VPPMsg/CancelRequestMsg.h"
class OCRTable : public Hashtable<RequestToken*, OutboundCallRequest*> {
public:
   OCRTable() : Hashtable<RequestToken*, OutboundCallRequest*>() {}
   virtual ~OCRTable() {
      clear();
   }
   int _hash(RequestToken* token) {
      return (int)token->GetRequestID();
   }
   RequestToken* _copy(RequestToken* value) {
      return new RequestToken(value->GetRequestID());
   }
   bool _keyEquals(RequestToken* k1, RequestToken* k2) {
      return (*k1) == (*k2);
   }
   bool _valueEquals(OutboundCallRequest* v1,
                     OutboundCallRequest* v2) {
      RequestToken rt1 = v1->GetRequestToken();
      RequestToken rt2 = v2->GetRequestToken();
      return rt1 == rt2;
   }
   void _deleteKey(RequestToken* key) {
      delete key;
   }
};
RequestToken*
Hashtable<RequestToken*,OutboundCallRequest*>::NULL_KEY = NULL;
OutboundCallRequest*
Hashtable<RequestToken*,OutboundCallRequest*>::NULL_VALUE = NULL;
Hashtable<RequestToken*, OutboundCallRequest*>*
OutboundCallRequest::table = new OCRTable();
Queue<OutboundCallRequest*>
OutboundCallRequest::zombieList;
OutboundCallRequest*
OutboundCallRequest::Create()
{
   OutboundCallRequest* ocq = new OutboundCallRequest();
   table->put(&ocq->GetRequestToken(), ocq);
```

```
  return ocq;
}
void
OutboundCallRequest::Destroy(OutboundCallRequest* r)
{
  table->remove(&r->GetRequestToken());
        ConnectionMgr::removeConnectionListener(r);
  zombieList.push(r);
  Synchronized sync(zombieList);
  if (zombieList.size() > 60) { // just a magic number.
    OutboundCallRequest* req = zombieList.pop();
    delete req;
  }
}
OutboundCallRequest*
OutboundCallRequest::GetOutboundCallRequest(RequestToken& token)
{
  return table->get(&token);
}
OutboundCallRequest::OutboundCallRequest()
  : hasResult(false), token(GetNextRequestID()), call(NULL), session(NULL)
{
}
OutboundCallRequest::~OutboundCallRequest()
{
}
bool
OutboundCallRequest::RemoveFromTable() {
  Synchronized sync(this);
  if (table->remove(&GetRequestToken())) {
    zombieList.push(this);
    return true;
  } else {
    return false;
  }
}
void
OutboundCallRequest::CallAlerting(invui::Call_ptr c) {
  Synchronized sync(this);
  status = CallProgressNotification::CPN_Alerting;
  call = c;
  hasResult = true;
  // RemoveFromTable();
  sync.notifyAll();
}
void
OutboundCallRequest::CallFailure(CallProgressNotification::CallProgressT s)
{
  Synchronized sync(this);
  status = s;
  hasResult = true;
  // RemoveFromTable();
  sync.notifyAll();
}
invui::Call_ptr
OutboundCallRequest::GetCall(long timeout) {
```

```
  Synchronized sync(this);
  while (hasResult == false && call == NULL) {
    sync.wait(timeout);
  }
  Destroy(this);
  if (call != NULL) {
    invui::Call_ptr c = call;
    call = NULL;
    VUISessionImpl* session = GetSession();
    session->SetCallAlerting();
    return c;
  }
  switch (status) {
  case CallProgressNotification::CPN_UserBusy:
    throw invui::UserBusy();
  case CallProgressNotification::CPN_NetworkBusy:
    throw invui::NetworkBusy();
  case CallProgressNotification::CPN_CallRejected:
    throw invui::CallRejected();
  case CallProgressNotification::CPN_NoAnswer:
    throw invui::Timeout();
  case CallProgressNotification::CPN_NoRingback:
    throw invui::NetworkError();
  case CallProgressNotification::CPN_NoDialtone:
    throw invui::InvalidNumber();
        case CallProgressNotification::CPN_InvalidNumber:
    throw invui::InvalidNumber();
        case CallProgressNotification::CPN_NetworkError:
    throw invui::NetworkError();
  case CallProgressNotification::CPN_Error:
    throw invui::TelephonyError(TSPF_SYSTEMFAILURE);
  default:
    // cancel call
    VUISessionImpl* session = GetSession();
    session->terminate();
    throw invui::Timeout();
  }
}
void
OutboundCallRequest::Disconnected() {
  Synchronized sync(this);
  GetSession();
  hasResult = true;
  sync.notifyAll();
  // when the call is not yet taken by anybody, cancel it.
  if (status == CallProgressNotification::CPN_Alerting && call != NULL) {
    delete call;
    call = NULL;
    CancelRequestMessage msg(token);
    session->HandleCancelRequest(&msg, NULL);
  }
}
REQUESTID
OutboundCallRequest::GetNextRequestID()
{
  static REQUESTID id = 0; // should be instance var.
```

```
    Synchronized sync(zombieList);
    id++;
    if (id == INVALID_REQUESTID) {
        id ++;
    }
    return id;
}
void
OutboundCallRequest::SetSession(VUISessionImpl* s) {
    Synchronized sync(this);
    session = s;
    sync.notifyAll();
}
VUISessionImpl*
OutboundCallRequest::GetSession() {
    Synchronized sync(this);
    while (session == NULL) {
        sync.wait();
    }
    return session;
}
```

Document Number:

```
/*
 * PlaceCallMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.12 $
 * $Date: 2000/12/07 03:21:51 $
 */
static const char* id="$Id: PlaceCallMsg.cpp,v 1.12 2000/12/07 03:21:51 Mitsuru Oshima Exp $";
#include "PlaceCallMsg.h"
#include "Property.h"
#include "PropertyList.h"
PlaceCallMessage::PlaceCallMessage(const PlaceCallMessage& other)
        : OneCallTokenMessageBase(other)
{
        cTarget = other.cTarget;
        cCallingNumber = other.cCallingNumber;
        cRequest = other.cRequest;
        cProperties = new PropertyList;
        *cProperties = *other.cProperties;
        cOwned = true;
        cQueue = other.cQueue;
}
//virtual
PlaceCallMessage::~PlaceCallMessage(void)
{
        if (cOwned)
                delete cProperties;
}
// virtual
void
PlaceCallMessage::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        cTarget.Serialize(this);
        cCallingNumber.Serialize(this);
        cRequest.Serialize(this);
        cProperties->Serialize(this);
        InsertUshort((unsigned short)cQueue);
        cCDRInfo.Serialize(this);
}
// virtual
void
PlaceCallMessage::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        cTarget.Deserialize(this);
        cCallingNumber.Deserialize(this);
        cRequest.Deserialize(this);
        cOwned = true;
        cProperties = new PropertyList;
        cProperties->Deserialize(this);
```

```
        cQueue = (bool)(ExtractUshort() != 0);
        cCDRInfo.Deserialize(this);
}
#if 0
// virtual
VirtualMachineID&
PlaceCallMessage::GetMTSVirtualMachineID(void)
{
        static VirtualMachineID invalid;
        if (cCallToken.IsValid())
                return cCallToken.GetMTSVirtualMachineID();
        return invalid;
}
#endif
```

```
/*
 * PlayMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:21:52 $
 */
static const char* id="$Id: PlayMsg.cpp,v 1.5 2000/12/07 03:21:52 Mitsuru Oshima Exp $";
#include "PlayMsg.h"
// virtual
void
PlayMessage::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        InsertString(cProductName);
        InsertString(cPromptName);
        InsertUshort(cLangID);
        InsertString(cTTSAlternate);
        InsertUshort(cRecordingID);
        InsertUshort((unsigned short)cUseSearchPath);
}
// virtual
void PlayMessage::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        ExtractString(cProductName);
        ExtractString(cPromptName);
        cLangID = (GMLANGID)ExtractUshort();
        ExtractString(cTTSAlternate);
        cRecordingID = (RECORDINGID)ExtractUshort();
        cUseSearchPath = (bool)(ExtractUshort() != 0);
}
```

```
/*
 * PromptMgr.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.7 $
 * $Date: 2000/12/07 03:19:49 $
 */
static const char* id="$Id: PromptMgr.cpp,v 1.7 2000/12/07 03:19:49 Mitsuru Oshima Exp $";
#include "PromptMgr.h"
#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <process.h>
#include <list>
#include "VPPUtils/GMLangMgr.h"
#include "VPPUtils/Synchronize.h"
#pragma warning(disable:4786)
gm::string PromptManager::cPromptRoot;
CriticalSection PromptManager::cCriticalSection;
bool PromptManager::cRandomizerInitialized = false;
const short PromptManager::MAXSEARCHPATH = 256;
void
PromptManager::SetPromptRoot(const gm::string& promptroot)
{
        Synchronize sync(cCriticalSection);
        cPromptRoot = promptroot;
        if (promptroot)
        {
                char c = promptroot[promptroot.length() - 1];
                if (c != '/' && c != '\\')
                        cPromptRoot += "/";
        }
}
bool
PromptManager::BuildPromptFilename(const gm::string& productname,
        const gm::string& basename, GMLANGID langID, gm::string& result)
{
        result.erase();

        if (!basename)
                return true;

        Synchronize sync(cCriticalSection);
        gm::string primary, sublang;
        if (GMLanguageManager::GMLANGIDToStrings(langID, primary, sublang))
        {
                if (cPromptRoot)
                        result = cPromptRoot;
                else
                        result = ".";
                result += productname;
```

```
                result += "/";
                result += primary;
                result += "/";
                result += sublang;
                result += "/prompts/";
                result += basename;
                return true;
        }
        return false;
}
bool
PromptManager::BuildPromptFilenameUsingSearchPath(const gm::string& searchpath,
                const gm::string& basename, gm::string& result)
{
        result.erase();

        if (!basename)
                return true;

        Synchronize sync(cCriticalSection);
        bool ret=true;
        char *str;
        if (!searchpath.c_str() ||
                !(*searchpath.c_str()))
        {
                // search path is not defined, go with the root prompt
                gm::string pathname = GetPromptRoot();
                pathname += basename;
                // test existence
                int res = _access(pathname, 0);
                if (res == 0) // file exists
                {
                        result = pathname;
                }
        }
        else
        {
                if (!(str = strdup(searchpath))) // somehow strdup fails
                        ret = false;
                else
                {
                        char *substr = 0;
                        while ((substr = strtok(substr ? 0 : str, ", \t;")))
                        {
                                gm::string root = GetPromptRoot();
                                gm::string pathname = substr;
                                // if pathname starts with a single / or \, attach current
                                // directory to it
                                // if pathname contains \\, //, :\, or :/, leave them alone
                                // if it does not, it means that it's a relative path
                                // and the prompt root needs to be prefix to it
                                if (!strstr(pathname.c_str(), "\\\\") &&
                                        !strstr(pathname.c_str(), "//") &&
                                        !strstr(pathname.c_str(), ":\\") &&
                                        !strstr(pathname.c_str(), ":/") &&
                                        !strstr(pathname.c_str(), ":")
```

```
                )
            {
                // example: \testdir - make it a full path
                if (pathname.c_str() &&
                    ((*pathname.c_str() == '\\') ||
                     (*pathname.c_str() == '/'))
                    )
                {
                    char fullpath[MAXSEARCHPATH],
relpath[MAXSEARCHPATH];
                    strncpy (relpath, pathname.c_str(), MAXSEARCHPATH-1);
                    if(_fullpath( fullpath, relpath, MAXSEARCHPATH) != NULL)
                    {
                        pathname = fullpath;
                    }
                    else
                        continue;
                }
                else  // this is the case that does not have \ or / as a prefix
                {
                    root += pathname;
                    pathname = root;
                }
            }
            // don't add a / if there is one already
            if (pathname.c_str() &&
                ((*(pathname.c_str()+strlen(pathname.c_str())-1)) != '/') &&
                ((*(pathname.c_str()+strlen(pathname.c_str())-1)) != '\\')
                )
                pathname += "/";
            pathname += basename;
            // test existence
            int res = _access(pathname, 0);
            if (res == 0) // found the file, stop searching
            {
                result = pathname;
                break;
            }
        }
        free(str);
        }
    }
    if (!result.c_str() || !(*result.c_str()))
        ret = false;
    return ret;
}
bool
PromptManager::GenerateUniqueFilename(const gm::string& subdir, gm::string& result)
{
    if (!cPromptRoot)
        return false;
    result = cPromptRoot;
    result += subdir;
    result += "/";
    return GenerateRandomUniqueFilename(result);
}
```

```
bool
PromptManager::GenerateUniqueFilename(const gm::string& productname,
        GMLANGID langID, const gm::string& subdir, gm::string& result)
{
        gm::string primary, sublang;
        if (GMLanguageManager::GMLANGIDToStrings(langID, primary, sublang))
        {
                if (cPromptRoot)
                        result = cPromptRoot;
                else
                        result = ".";
                result += productname;
                result += "/";
                result += primary;
                result += "/";
                result += sublang;
                result += "/";
                result += subdir;
                result += "/";
                return GenerateRandomUniqueFilename(result);
        }
        return false;
}
bool
PromptManager::GenerateRandomUniqueFilename(gm::string& result)
{
        Synchronize sync(cCriticalSection);
        char filename[20];
        if (!cRandomizerInitialized)
        {
                srand(_getpid());
                cRandomizerInitialized = true;
        }
        while (true)
        {
                int r = rand();
                itoa(r, filename, 10);
                gm::string fn = filename;
                fn += ".wav";
                gm::string str = result.c_str();
                str += fn;
                FILE *fp = fopen(str, "rb");
                if (!fp)
                {
                        if (result.c_str() && !strcmp(result.c_str(), ""))
                                result = fn;
                        else
                                result += fn;
                        break;
                }
                fclose(fp);
        }
        return true;
}
bool
PromptManager::BuildRecordingFilename(const gm::string& productname,
```

```
        GMLANGID langID, gm::string& result)
{
        return GenerateUniqueFilename(productname, langID, "recordings", result);
}
bool
PromptManager::BuildTTSFilename(const gm::string& productname, GMLANGID langID,
        gm::string& result)
{
        return GenerateUniqueFilename(productname, langID, "tts", result);
}
bool
PromptManager::BuildTTSFilename(gm::string& result)
{
        return GenerateUniqueFilename("tts", result);
}
```

Document ID: 20030826.0

```
/*
 * PromptSegment.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:19:49 $
 */
static const char* id="$Id: PromptSegment.cpp,v 1.2 2000/12/07 03:19:49 Mitsuru Oshima Exp $";
#include "PromptSegment.h"
// virtual
PromptSegment::~PromptSegment(void)
{
}
```

Document ID: 20020702-0000

```
/*
 * PromptSegmentMgr.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:19:50 $
 */
static const char* id="$Id: PromptSegmentMgr.cpp,v 1.4 2000/12/07 03:19:50 Mitsuru Oshima Exp $";
#include "PromptSegmentMgr.h"
#include "PromptSegment.h"
PromptSegmentManager::PromptSegmentManager(void)
{
}
PromptSegmentManager::~PromptSegmentManager(void)
{
        Clear();
}
void
PromptSegmentManager::AddSegment(PromptSegment *pSegment)
{
        cList.push_back(pSegment);
}
void
PromptSegmentManager::Clear(void)
{
        PromptSegment *pSeg = GetFirst();
        while (pSeg)
        {
                PromptSegment *pNext = GetNext();
                delete pSeg;
                pSeg = pNext;
        }

        cList.clear();
}
```

```
/*
 * Property.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:21:52 $
 */
static const char* id="$Id: Property.cpp,v 1.4 2000/12/07 03:21:52 Mitsuru Oshima Exp $";
#include <stdlib.h>
#include "Property.h"
#include "VPPMsg.h"
Property::Property(const gm::string& name, long value)
{
        Setup(name, value);
}
void
Property::Setup(const gm::string& name, long value)
{
        cName = name.c_str();
        char buf[34];
        ::_ltoa(value, buf, 10);
        cValue = buf;
}
long
Property::GetLongValue(void)
{
        return atol(cValue);
}
void
Property::Serialize(VPPMessage *pMsg)
{
        pMsg->InsertString(cName);
        pMsg->InsertString(cValue);
}
void
Property::Deserialize(VPPMessage *pMsg)
{
        pMsg->ExtractString(cName);
        pMsg->ExtractString(cValue);
}
```

```
/*
 * PropertyList.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.8 $
 * $Date: 2000/12/07 03:21:52 $
 */
static const char* id="$Id: PropertyList.cpp,v 1.8 2000/12/07 03:21:52 Mitsuru Oshima Exp $";
#include <stdlib.h>
#include "PropertyList.h"
#include "Property.h"
#include "VPPMsg.h"
// virtual
PropertyList::~PropertyList(void)
{
        std::list<Property *>::iterator iter = cProperties.begin();
        while (iter != cProperties.end())
        {
                delete *iter;
                iter++;
        }
}
bool
PropertyList::RemoveProperty(const gm::string& name)
{
        std::list<Property *>::iterator iter = cProperties.begin();
        while (iter != cProperties.end())
        {
                if ((*iter)->GetName() == name)
                {
                        delete *iter;
                        cProperties.erase(iter);
                        return true;
                }
                iter++;
        }
        return false;
}
bool
PropertyList::SetProperty(const gm::string& name, const gm::string& value)
{
        RemoveProperty(name);
        cProperties.push_back(new Property(name, value));
        return true;
}
bool
PropertyList::SetProperty(const gm::string& name, long value)
{
        RemoveProperty(name);
        char buf[10];
        ::ltoa(value, buf, 10);
```

```
        cProperties.push_back(new Property(name, buf));
        return true;
}
bool
PropertyList::GetProperty(const gm::string& name, gm::string& value)
{
        std::list<Property *>::iterator iter = cProperties.begin();
        while (iter != cProperties.end())
        {
                if ((*iter)->GetName() == name)
                {
                        value = (*iter)->GetValue();
                        return true;
                }
                iter++;
        }
        return false;
}
bool
PropertyList::GetProperty(const gm::string& name, long& value)
{
        std::list<Property *>::iterator iter = cProperties.begin();
        while (iter != cProperties.end())
        {
                if ((*iter)->GetName() == name)
                {
                        value = ::atol((*iter)->GetValue());
                        return true;
                }
                iter++;
        }
        return false;
}
void
PropertyList::Serialize(VPPMessage *pMsg)
{
        pMsg->InsertUshort((unsigned short)cProperties.size());
        std::list<Property *>::iterator iter = cProperties.begin();
        while (iter != cProperties.end())
        {
                (*iter)->Serialize(pMsg);
                iter++;
        }
}
void
PropertyList::Deserialize(VPPMessage *pMsg)
{
        unsigned short cnt = pMsg->ExtractUshort();
        for (unsigned short i = 0; i < cnt; i++)
        {
                gm::string name, value;
                pMsg->ExtractString(name);
                pMsg->ExtractString(value);
                SetProperty(name, value);
        }
}
```

```
void
PropertyList::Merge(const PropertyList& other, bool EraseFirst)
{
	std::list<Property *>::const_iterator iter = other.cProperties.begin();
	if (EraseFirst)
		cProperties.clear();
	while (iter != other.cProperties.end())
	{
		SetProperty((*iter)->GetName(), (*iter)->GetValue());
		iter++;
	}
}
PropertyList&
PropertyList::operator=(const PropertyList& other)
{
	Merge(other, true);
	return *this;
}
```

```
/*
 * PropertyMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:53 $
 */
static const char* id="$Id: PropertyMsg.cpp,v 1.3 2000/12/07 03:21:53 Mitsuru Oshima Exp $";
#include "PropertyMsg.h"
// virtual
void
PropertyMessage::Serialize(void)
{
        VPPMessage::Serialize();
        cProperty.Serialize(this);
}
// virtual
void
PropertyMessage::Deserialize(void)
{
        VPPMessage::Deserialize();
        cProperty.Deserialize(this);
}
```

```
/*
 * QueryDynamicGrammarContentsResultNtfy.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:53 $
 */
static const char* id="$Id: QueryDynamicGrammarContentsResultNtfy.cpp,v 1.2 2000/12/07 03:21:53 Mitsuru
Oshima Exp $";
#include "QueryDynamicGrammarContentsResultNtfy.h"
// virtual
void
QueryDynamicGrammarContentsResultNotification::Serialize(void)
{
        DynamicGrammarPhraseResultNotification::Serialize();
        InsertString(cPhraseContents);
}
// virtual
void
QueryDynamicGrammarContentsResultNotification::Deserialize(void)
{
        DynamicGrammarPhraseResultNotification::Deserialize();
        ExtractString(cPhraseContents);
}
```

```
/*
 * RangeParser.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.6 $
 * $Date: 2000/12/07 03:19:50 $
 */
static const char* id="$Id: RangeParser.cpp,v 1.6 2000/12/07 03:19:50 Mitsuru Oshima Exp $";
#include <stdlib.h>
#include <memory.h>
#include "RangeParser.h"
RangeParser::RangeParser(const gm::string& str)
{
        cMaxValue = Parse(str, false);
        int numbytes = ((cMaxValue + 7) >> 3);
        if (!numbytes)
                numbytes = 1;
        cBits = new unsigned char [numbytes];
        ::memset(cBits, 0, numbytes);
        Parse(str, true);
}
RangeParser::~RangeParser(void)
{
        delete cBits;
}
unsigned short
RangeParser::Parse(const gm::string& str, bool doit)
{
        const char *ptr = str;
        gm::string tmp;
        int val, val2;
        unsigned short maxval = 0;
        maxval = 0;
        while (ptr && *ptr)
        {
                tmp.erase();
                while (*ptr && *ptr != '-' && *ptr != ',')
                        tmp += *ptr++;
                val = ::atoi(tmp.c_str());
                if (val > maxval)
                        maxval = val;
                if (*ptr == '-')
                {
                        // This is a range
                        //
                        ptr++;
                        tmp.erase();
                        while (*ptr && *ptr != ',')
                                tmp += *ptr++;
                        if (*ptr == ',')
                                ptr++;
```

```
                        val2 = ::atoi(tmp.c_str());
                        if (val2 > maxval)
                                maxval = val2;
                        if (doit)
                        {
                                for (unsigned short i = val; i <= val2; i++)
                                        SetBit(i);
                        }
                }
                else
                {
                        if (*ptr)
                                ptr++;
                        if (doit)
                                SetBit(val);
                }
        }
        return maxval;
}
void
RangeParser::SetBit(unsigned short ndx)
{
        if (ndx <= cMaxValue)
                cBits[(ndx >> 3)] |= (1 << (7 - (ndx % 8)));
}
bool
RangeParser::IsSet(unsigned short ndx)
{
        if (ndx <= cMaxValue)
                return (bool)(((cBits[(ndx >> 3)]) >> (7 - (ndx % 8))) & 0x1);
        else
                return false;
}
```

```
/*
 * RecognitionDoneNtfy.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:21:53 $
 */
static const char* id="$Id: RecognitionDoneNtfy.cpp,v 1.5 2000/12/07 03:21:53 Mitsuru Oshima Exp $";
#include "RecognitionDoneNtfy.h"
#include "RecognitionResults.h"
#include "RecognitionResult.h"
#include "Slot.h"
// virtual
RecognitionDoneNotification::~RecognitionDoneNotification(void)
{
        if (cOwned)
                delete cResults;
}
// virtual
void
RecognitionDoneNotification::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        InsertUshort((unsigned short)(cResults ? 1 : 0));
        if (cResults)
                cResults->Serialize(this);
        InsertUshort((unsigned short)cStatus);
}
// virtual
void
RecognitionDoneNotification::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        cOwned = true;
        unsigned short flag = ExtractUshort();
        if (flag)
        {
                cResults = new RecognitionResults;
                cResults->Deserialize(this);
        }
        else
                cResults = 0;
        cStatus = (RecognitionStatus)ExtractUshort();
}
```

```
/*
 * RecognitionResult.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.7 $
 * $Date: 2000/12/07 03:21:54 $
 */
static const char* id="$Id: RecognitionResult.cpp,v 1.7 2000/12/07 03:21:54 Mitsuru Oshima Exp $";
#include "RecognitionResult.h"
#include "NLInterpretation.h"
#include "VPPMsg.h"
RecognitionResult::RecognitionResult(void)
{
        cConfidence = 0.0;
}
RecognitionResult::~RecognitionResult(void)
{
        std::list<NLInterpretation *>::iterator iter =  cNLInterpretations.begin();
        while (iter != cNLInterpretations.end())
        {
                NLInterpretation *pNLInterpretation = *iter;
                delete pNLInterpretation;
                iter++;
        }
}
unsigned short
RecognitionResult::GetNumNLInterpretations(void)
{
        return (unsigned short)cNLInterpretations.size();
}
NLInterpretation *
RecognitionResult::GetNLInterpretation(int index)
{
        std::list<NLInterpretation *>::iterator iter =  cNLInterpretations.begin();
        int i = 0;
        while (iter != cNLInterpretations.end() && i < index)
        {
                i++;
                iter++;
        }
        if (iter != cNLInterpretations.end())
                return *iter;
        return 0;
}
void
RecognitionResult::AddNLInterpretation(NLInterpretation *pNLInterpretation)
{
        cNLInterpretations.push_back(pNLInterpretation);
}
void
RecognitionResult::Serialize(VPPMessage *pMsg)
```

```
{
        pMsg->InsertString(cResult);
//      pMsg->InsertFloat(cConfidence);
        pMsg->InsertUshort(GetNumNLInterpretations());
        std::list<NLInterpretation *>::iterator iter =  cNLInterpretations.begin();
        while (iter != cNLInterpretations.end())
        {
                (*iter)->Serialize(pMsg);
                iter++;
        }
}
void
RecognitionResult::Deserialize(VPPMessage *pMsg)
{
        pMsg->ExtractString(cResult);
//      cConfidence = pMsg->ExtractFloat();
   cConfidence = 1.0;
        unsigned short num = pMsg->ExtractUshort();
        cNLInterpretations.clear();
        for (unsigned short i = 0; i < num; i++)
        {
                NLInterpretation *pNLInterpretation = new NLInterpretation;
                pNLInterpretation->Deserialize(pMsg);
                AddNLInterpretation(pNLInterpretation);
        }
}
```

```
/*
 * RecognitionResults.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:21:54 $
 */
static const char* id="$Id: RecognitionResults.cpp,v 1.4 2000/12/07 03:21:54 Mitsuru Oshima Exp $";
#include "RecognitionResults.h"
#include "RecognitionResult.h"
#include "Slot.h"
#include "VPPMsg.h"
RecognitionResults::RecognitionResults(void)
{
}
RecognitionResults::~RecognitionResults(void)
{
        std::list<RecognitionResult *>::iterator iter = cResults.begin();
        while (iter != cResults.end())
        {
                RecognitionResult *pResult = *iter;
                delete pResult;
                iter++;
        }
}
unsigned short
RecognitionResults::GetNumResults(void)
{
        return cResults.size();
}
RecognitionResult *
RecognitionResults::GetResult(int index)
{
        int i = 0;
        std::list<RecognitionResult *>::iterator iter = cResults.begin();
        while (iter != cResults.end() && i < index)
        {
                i++;
                iter++;
        }
        if (iter != cResults.end())
                return *iter;
        return 0;
}
void
RecognitionResults::AddResult(RecognitionResult *pResult)
{
        cResults.push_back(pResult);
}
void
RecognitionResults::Serialize(VPPMessage *pMsg)
```

```
{
        pMsg->InsertUshort(GetNumResults());
        std::list<RecognitionResult *>::iterator iter = cResults.begin();
        while (iter != cResults.end())
        {
                (*iter)->Serialize(pMsg);
                iter++;
        }
}
void
RecognitionResults::Deserialize(VPPMessage *pMsg)
{
        unsigned short num = pMsg->ExtractUshort();
        for (unsigned short i = 0; i < num; i++)
        {
                RecognitionResult *pResult = new RecognitionResult;
                pResult->Deserialize(pMsg);
                AddResult(pResult);
        }
}
```

```
/*
 * RecognizeStringMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:55 $
 */
static const char* id="$Id: RecognizeStringMsg.cpp,v 1.2 2000/12/07 03:21:55 Mitsuru Oshima Exp $";
#include "RecognizeStringMsg.h"
// virtual
void
RecognizeStringMessage::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        InsertString(cGrammar);
        InsertString(cStr);
}
// virtual
void RecognizeStringMessage::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        ExtractString(cGrammar);
        ExtractString(cStr);
}
```

```
/*
 * RecordingDoneNtfy.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:55 $
 */
static const char* id="$Id: RecordingDoneNtfy.cpp,v 1.2 2000/12/07 03:21:55 Mitsuru Oshima Exp $";
#include "RecordingDoneNtfy.h"
//virtual
void RecordingDoneNotification::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
  InsertUshort((unsigned short)cReason);
}
//virtual
void RecordingDoneNotification::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
  cReason = (RecordingDoneReason)ExtractUshort();
}
```

```
/*
 * RecordingHandedOffNtfy.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:55 $
 */
static const char* id="$Id: RecordingHandedOffNtfy.cpp,v 1.3 2000/12/07 03:21:55 Mitsuru Oshima Exp $";
#include "RecordingHandedOffNtfy.h"
// virtual
void
RecordingHandedOffNotification::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        InsertUshort(cID);
}
// virtual
void
RecordingHandedOffNotification::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        cID = (RECORDINGID)ExtractUshort();
}
```

```
/*
 * RecordingHandler.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:19:50 $
 */
static const char* id="$Id: RecordingHandler.cpp,v 1.2 2000/12/07 03:19:50 Mitsuru Oshima Exp $";
#include "RecordingHandler.h"
// virtual
RecordingHandler::~RecordingHandler(void)
{
}
```

0 6 0 2 6 0 2 6 0 2

```
/*
 * RecordingMgr.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:19:50 $
 */
static const char* id="$Id: RecordingMgr.cpp,v 1.4 2000/12/07 03:19:50 Mitsuru Oshima Exp $";
#include "RecordingMgr.h"
#include "VPPUtils/Synchronize.h"
RECORDINGID RecordingManager::cNextRecordingID = 0;
CriticalSection RecordingManager::cCriticalSection;
RecordingManager::~RecordingManager(void)
{
        std::list<RecordingNode *>::iterator iter = cRecordings.begin();
        while (iter != cRecordings.end())
        {
                delete *iter;
                iter++;
        }
}
RECORDINGID
RecordingManager::Add(gm::string& filename)
{
        Synchronize sync(cCriticalSection);
        RecordingNode *pNode = new RecordingNode;
        pNode->cRecordingID = cNextRecordingID++;
        pNode->cFilename = filename;
        cRecordings.push_back(pNode);

        if (cNextRecordingID == INVALID_RECORDINGID)
                cNextRecordingID++;
        return pNode->cRecordingID;
}
bool
RecordingManager::Remove(RecordingNode *pNode)
{
        cRecordings.remove(pNode);
        delete pNode;
        return true;
}
bool
RecordingManager::Remove(gm::string& filename)
{
        std::list<RecordingNode *>::iterator iter = cRecordings.begin();
        while (iter != cRecordings.end() && (*iter)->cFilename != filename)
                iter++;
        if (iter != cRecordings.end())
                return Remove(*iter);
        return false;
}
```

```
bool
RecordingManager::Remove(RECORDINGID recordingID)
{
	std::list<RecordingNode *>::iterator iter = cRecordings.begin();
	while (iter != cRecordings.end() && (*iter)->cRecordingID != recordingID)
		iter++;
	if (iter != cRecordings.end())
		return Remove(*iter);
	return false;
}
RECORDINGID
RecordingManager::Lookup(gm::string& filename)
{
	std::list<RecordingNode *>::iterator iter = cRecordings.begin();
	while (iter != cRecordings.end() && (*iter)->cFilename != filename)
		iter++;
	if (iter != cRecordings.end())
		return (*iter)->cRecordingID;
	return INVALID_RECORDINGID;
}
bool
RecordingManager::Lookup(RECORDINGID recordingID, gm::string& filename)
{
	std::list<RecordingNode *>::iterator iter = cRecordings.begin();
	while (iter != cRecordings.end() && (*iter)->cRecordingID != recordingID)
		iter++;
	if (iter != cRecordings.end())
	{
		filename = (*iter)->cFilename;
		return true;
	}
	return false;
}
int
RecordingManager::GetNumEntries(void)
{
	return cRecordings.size();
}
bool
RecordingManager::GetFilename(int index, gm::string& filename)
{
	std::list<RecordingNode *>::iterator iter = cRecordings.begin();
	int i = 0;
	while (i < index && iter != cRecordings.end())
	{
		i++;
		iter++;
	}
	if (iter != cRecordings.end())
	{
		filename = (*iter)->cFilename;
		return true;
	}
	return false;
}
```

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <time.h>
#include <sys/timeb.h>
#include "CMS/CMSClient.h"
//#include "CMS/IPCPipe.h"
#include "CMS/IPCSocket.h"
#include "VPPUtils/CmdLine.h"
#include "VPPUtils/VPPDebug.h"
#include "VPPUtils/GMString"
#define TESTMSGS 1000
int num;
DWORD times[TESTMSGS];
HANDLE ReplyReceivedEvent;
bool
PingFailureMessageHandler(CMSMessage *pMsg, CMSEndpoint *pEndpoint,
        void *userdata)
{
        printf("Server not responding to ping!\n");
        return true;
}
DWORD
GetTimestamp(void)
{
        struct _timeb tb;
        _ftime(&tb);
        struct tm *time = localtime(&tb.time);
        return (DWORD)tb.millitm +
                ((DWORD)time->tm_sec * 1000L) +
                ((DWORD)time->tm_min * 60000L) +
                ((DWORD)time->tm_hour * 3600000L);
}
bool
TimerMessageHandler(CMSMessage *pMsg, CMSEndpoint *pEndpoint,
        void *userdata)
{
        DWORD tc = GetTimestamp();
        times[num++] = tc - pMsg->GetUserdata();
        ::SetEvent(ReplyReceivedEvent);
        printf("Got reply %d at %ld with start %ld, delta %ld\n", num, tc, pMsg->GetUserdata(),
                tc - pMsg->GetUserdata());

        if (num == TESTMSGS)
        {
                DWORD total = 0;
                for (int i = 0; i < TESTMSGS; i++)
                {
//                      printf("times[%d] = %ld, ", i, times[i]);
                        total += times[i];
                }
                DWORD avg = total / TESTMSGS;
                printf("\nTotal of %d messages was %ld ticks\n", TESTMSGS, total);
                printf("\nRound-trip average of %d messages was %ld ticks\n", TESTMSGS, avg);
                fflush(stdout);
```

```
        }
        return true;
}
void
main(int ac, char **av)
{
        char machinename[100];
        gm::string logname = "c:\\testclient.";
        char buf[20];
        itoa(_getpid(), buf, 16);
        logname += buf;
        logname += ".out";
        debug::SetTeeOutput(logname);
        CommandLine cmdline(ac, av);
        gm::string value = cmdline.GetValue("MachineName");
        if (value)
                strcpy(machinename, value);
        else
                strcpy(machinename, ".");
        printf("Attempting to connect to server on %s\n", machinename);
#ifdef USEPIPE
        IPCPipe *pEndpoint = new IPCPipe(machinename, "testpipe");
#else
        IPCSocket *pEndpoint = new IPCSocket(machinename, 1234);
#endif
        CMSClient client(pEndpoint);
        client.AddMessageHandler(0xaaaa, TimerMessageHandler, 0);
        client.AddMessageHandler(C_NTFY_PINGFAILURE, PingFailureMessageHandler, 0);

        if (client.Connect())
        {
                unsigned short id = 0xdfb;
                CMSMessage msg, recvmsg;
                while (client.IsConnected())
                {
                        msg.SetCommandID(id);
                        printf("SPACE = send one msg., C = send 64K msgs.\n");
                        printf("T = time msg, ESC = quit\n");
                        int ch = getch();
                        int nummsgs;
                        bool TimeIt = false;
                        if (ch == ' ')
                                nummsgs = 1;
                        else if (ch == 'C' || ch == 'c')
                                nummsgs = 0xffff;
                        else if (ch == 'T' || ch == 't')
                        {
                                nummsgs = 1;
                                TimeIt = true;
                                msg.SetCommandID(0xaaaa);
                        }
                        else if (ch == 27)
                                break;
                        else
                                nummsgs = 0;
                        while (client.IsConnected() && nummsgs--)
```

```
                {
                        if (TimeIt)
                        {
                                ReplyReceivedEvent = ::CreateEvent(NULL, TRUE, FALSE, NULL);
                                DWORD StartTime = GetTimestamp();
                                num = 0;
                                for (int i = 0; i < TESTMSGS; i++)
                                {
                                        msg.SetUserdata(GetTimestamp());
                                        if (!client.WriteMessage(msg))
                                                printf("Client couldn't write message.\n");
                                        ::WaitForSingleObject(ReplyReceivedEvent, INFINITE);
                                        ::ResetEvent(ReplyReceivedEvent);
                                }
                                DWORD EndTime = GetTimestamp();
                                printf("\nTotal unit time was %ld ticks, average %ld\n",
                                        EndTime - StartTime, ((EndTime - StartTime) / TESTMSGS));
                        }
                        else if (!client.WriteMessage(msg))
                                printf("Client couldn't write message.\n");
                        else
                                printf("%d: Client wrote message ok.\n", msg.GetSerialNo());
                        if (kbhit())
                                break;
                        /*
                        while (client.MessagesWaiting())
                        {
                                CMSMessage *pMsg = client.GetNextMessage();
                                printf("Got message %d (%d)\n", pMsg->GetCommandID(),
                                        pMsg->GetSerialNo());
                                delete pMsg;
                        }
                        */
                }
        }
        client.Disconnect();
    }
}
```

```
/*
 * TTSChannel.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.19 $
 * $Date: 2000/12/07 03:19:54 $
 */
static const char* id="$Id: TTSChannel.cpp,v 1.19 2000/12/07 03:19:54 Mitsuru Oshima Exp $";
#include <assert.h>
#include <windows.h>
#include <process.h>
#include <stdio.h>
#include "VPPUtils/Log.h"
#include "VPPUtils/Synchronize.h"
#include "TTSChannel.h"
#include "SessionMgr.h"
static void _TTSChannelThread(void *);
TTSChannel::TTSChannel(ServiceProvider *pServiceProvider,
        const char *logType, int id, bool RequiresSettlement)
  : Channel(pServiceProvider, Channel::CT_TTS, logType, id, RequiresSettlement),
  m_State(TTS_Invalid),
  m_pString(NULL)
{
}
bool
TTSChannel::Shutdown(void)
{
        return true;
}
TTSChannel::~TTSChannel()
{
        if(m_pString)
        {
                delete [] m_pString;
        }
}
bool TTSChannel::InitializeSynthesis(const char *TTSstring)
{
        if (!TTSstring || !strlen(TTSstring)) {
                warning("TTS is requested for an empty or null string.");
                return false;
        }
        info("Start TTS for string");
        diag("String: %s", TTSstring);
        Synchronized sync(m_StateMonitor);
        if (m_State != TTS_Idle)
        {
                info("Waiting for previous TTS to stop");
                sync.wait(1000); // Wait a second or till idle
                if (m_State != TTS_Idle)
                {
```

```
                    warning("Trying to stop previous TTS");
                    StopTTS();
                    sync.wait(3000); // Wait a couple seconds or till idle
                }
        }
        if (m_State == TTS_Idle)
        {
                if(m_pString)
                {
                        delete [] m_pString;
                }
                m_pString = new char[strlen(TTSstring)+1];
                strcpy(m_pString, TTSstring);

                m_State = TTS_Initialized;
                return(StartTTS());
        }
        else
        {
                error("Couldn't stop previous TTS");
                return false;
        }
}
int TTSChannel::GetNextSamples(char *buffer, unsigned long bufferSize)
{
        int retLen = bufferSize;
        if(m_State == TTS_Initialized)
        {
                int dataSize;
                switch(SynthesisStatus())
                {
                case TTSSynthStatus_NotReady:
                        memset(buffer, '\xFF', bufferSize);
                        debug("No bytes available -- returning silence");
                        break;
                case TTSSynthStatus_Ready:
                        dataSize = GetNextBuffer(buffer, bufferSize);
                        if(dataSize < bufferSize)
                        {
                                memset(buffer+dataSize, '\xFF', bufferSize - dataSize);
                                debug("%d bytes are available -- returning silence", dataSize);
                        }
                        break;
                case TTSSynthStatus_EOD:
                        retLen = 0;
                        break;
                default:
                        retLen = 0;
                        break;
                }
        }
        else
        {
                error("GetNextSamples(), state is not TTS_Initialized");
                retLen = 0;
        }
```

```
        return retLen;
}
void TTSChannel::Release(void)
{
        diag("Releasing TTS");
        {
                Synchronized sync(m_StateMonitor);
                m_State = TTS_Stop;
        }
        StopTTS();
}
// virtual
void
TTSChannel::NotifyBoundToSession(SessionNode *pSessionNode)
{
        Channel::NotifyBoundToSession(pSessionNode);
        if (!pSessionNode)
                NotifySettled();
}
void
TTSChannel::NotifySettled(void)
{
        info("TTSChannel::NotifySettled, m_State = %d", m_State);
        Synchronized sync(SessionMgr::GetSessionMgr());
   Channel::NotifySettled();
        sync.notifyAll();
}
```

```
/*
 * RecordingNtfy.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:21:56 $
 */
static const char* id="$Id: RecordingNtfy.cpp,v 1.4 2000/12/07 03:21:56 Mitsuru Oshima Exp $";
#include "RecordingNtfy.h"
// virtual
void
RecordingNotification::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        InsertUshort(cID);
        InsertString(cRecordingFilename);
}
// virtual
void
RecordingNotification::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        cID = (RECORDINGID)ExtractUshort();
        ExtractString(cRecordingFilename);
}
const gm::string&
RecordingNotification::GetRecordingFilename(void)
{
        return cRecordingFilename;
}
```

```
/*
 * RecordMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:21:55 $
 */
static const char* id="$Id: RecordMsg.cpp,v 1.5 2000/12/07 03:21:55 Mitsuru Oshima Exp $";
#include "RecordMsg.h"
// virtual
void
RecordMessage::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        InsertString(cFileName);
  InsertUlong(cBosTimeout);
  InsertUlong(cTmsTimeout);
  InsertUlong(cEosTimeout);
}
// virtual
void RecordMessage::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        ExtractString(cFileName);
  cBosTimeout = ExtractUlong();
  cTmsTimeout = ExtractUlong();
  cEosTimeout = ExtractUlong();
}
```

```
/*
 * RegistryMgr.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.6 $
 * $Date: 2000/12/07 03:22:19 $
 */
static const char* id="$Id: RegistryMgr.cpp,v 1.6 2000/12/07 03:22:19 Mitsuru Oshima Exp $";
#include <windows.h>
#include "RegistryMgr.h"
bool
RegistryMgr::GetString(const char *parent, const char *key, gm::string& retString)
{
#if _MSC_VER < 1200
        HANDLE regKeyHndl;
#else
        HKEY__ *regKeyHndl;
#endif
    void *regKey = NULL;
        bool result = false;
    long rc;
        rc = RegOpenKeyEx(HKEY_LOCAL_MACHINE, parent, 0, KEY_READ, &regKeyHndl);
    if (rc != ERROR_SUCCESS) {
                return false;
        }
        unsigned long type, size;
        char *value;

        size = 0;
        rc = RegQueryValueEx(regKeyHndl, key, NULL, &type, NULL, &size);
        if ((rc == ERROR_SUCCESS) && (type == REG_SZ)) {
                value = new char[size];
                if (value != NULL) {
                        rc = RegQueryValueEx(regKeyHndl, key, NULL, NULL,
                                reinterpret_cast<unsigned char *>(value),
                                &size);
                        if (rc == ERROR_SUCCESS) {
                                if ((value != NULL) && (*value != '\0'))
                                {
                                        retString = value;
                                }
                                else
                                {
                                        retString = "";
                                        retString.resize(1, '\0');      // Forcing retString's char ptr to be non-NULL.
                                }
                                delete[] value;
                                result = true;
                        } else {
                                delete[] value;
                                result = false;
```

```
                    }
                }
            }
            RegCloseKey(regKeyHndl);
            return result;
}
```

Document ID: 1

```
/*
 * RegularExpression.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:33:44 $
 */
static const char* id="$Id: RegularExpression.cpp,v 1.2 2000/12/07 03:33:44 Mitsuru Oshima Exp $";
#include <string.h>
#include <malloc.h>
#include "RegularExpression.h"
#include "config.h"
#include "regex.h"
RegularExpression::RegularExpression(const char *str)
{
        cPatternBuffer = new struct re_pattern_buffer;
        cPatternBuffer->buffer = 0;
        cPatternBuffer->fastmap = 0;
        cPatternBuffer->translate = 0;
        setString(str);
}
RegularExpression::~RegularExpression(void)
{
        freeStrings();
        delete cPatternBuffer;
}
void
RegularExpression::freeStrings(void)
{
        if (cPatternBuffer->buffer)
        {
                ::free(cPatternBuffer->buffer);
                cPatternBuffer->buffer = 0;
        }
        if (cPatternBuffer->fastmap)
        {
                ::free(cPatternBuffer->fastmap);
                cPatternBuffer->fastmap = 0;
        }
        if (cPatternBuffer->translate)
        {
                ::free(cPatternBuffer->translate);
                cPatternBuffer->translate = 0;
        }
}
const char *
RegularExpression::setString(const char *str)
{
        freeStrings();
        if (str)
        {
```

```
                cPatternBuffer->buffer = (unsigned char *)::malloc(32);
                cPatternBuffer->allocated = 32;
                return ::re_compile_pattern(str, strlen(str), cPatternBuffer);
        }
        else
                return 0;
}
int
RegularExpression::match(const char *str, int startpos)
{
        if (cPatternBuffer->buffer)
                return ::re_match(cPatternBuffer, str, strlen(str), startpos, 0);
        else
                return -1;
}
int
RegularExpression::search(const char *str, int startpos, int range)
{
        if (cPatternBuffer->buffer)
                return ::re_search(cPatternBuffer, str, strlen(str), startpos, range, 0);
        else
                return -1;
}
```

```
/*
 * RemovePhraseFromDynamicGrammar.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:21:56 $
 */
static const char* id="$Id: RemovePhraseFromDynamicGrammar.cpp,v 1.2 2000/12/07 03:21:56 Mitsuru Oshima Exp
$";
```

Document ID: 1

```
/*
 * RemoveRecordingMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:56 $
 */
static const char* id="$Id: RemoveRecordingMsg.cpp,v 1.3 2000/12/07 03:21:56 Mitsuru Oshima Exp $";
#include "RemoveRecordingMsg.h"
// virtual
void
RemoveRecordingMessage::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        InsertUshort(cID);
}
// virtual
void
RemoveRecordingMessage::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        cID = (RECORDINGID)ExtractUshort();
}
```

```
/*
 * RequestCancelledNtfy.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:21:56 $
 */
static const char* id="$Id: RequestCancelledNtfy.cpp,v 1.5 2000/12/07 03:21:56 Mitsuru Oshima Exp $";
#include "RequestCancelledNtfy.h"
// virtual
void
RequestCancelledNotification::Serialize(void)
{
        VPPMessage::Serialize();
        cRequest.Serialize(this);
}
// virtual
void
RequestCancelledNotification::Deserialize(void)
{
        VPPMessage::Deserialize();
        cRequest.Deserialize(this);
}
```

```
/*
 * RequestToken.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:21:57 $
 */
static const char* id="$Id: RequestToken.cpp,v 1.5 2000/12/07 03:21:57 Mitsuru Oshima Exp $";
#include "RequestToken.h"
#include "VPPMsg.h"

void
RequestToken::Serialize(VPPMessage *pMsg)
{
#if 0
        cClientID.Serialize(pMsg);
        cMTSVirtualMachineID.Serialize(pMsg);
#endif
        pMsg->InsertUshort((unsigned short)cRequestID);
}
void
RequestToken::Deserialize(VPPMessage *pMsg)
{
#if 0
        cClientID.Deserialize(pMsg);
        cMTSVirtualMachineID.Deserialize(pMsg);
#endif
        cRequestID = (REQUESTID)pMsg->ExtractUshort();
}
```

```
/*
 * RequestTokenMsgBase.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:57 $
 */
static const char* id="$Id: RequestTokenMsgBase.cpp,v 1.3 2000/12/07 03:21:57 Mitsuru Oshima Exp $";
#include "RequestTokenMsgBase.h"
// virtual
void
RequestTokenMessageBase::Serialize(void)
{
        VPPMessage::Serialize();
        cRequestToken.Serialize(this);
}
// virtual
void
RequestTokenMessageBase::Deserialize(void)
{
        VPPMessage::Deserialize();
        cRequestToken.Deserialize(this);
}
#if 0
// virtual
VirtualMachineID&
RequestTokenMessageBase::GetMTSVirtualMachineID(void)
{
        return cRequestToken.GetMTSVirtualMachineID();
}
#endif
```

```
/*
 * SampleSegment.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.6 $
 * $Date: 2000/12/07 03:19:51 $
 */
static const char* id="$Id: SampleSegment.cpp,v 1.6 2000/12/07 03:19:51 Mitsuru Oshima Exp $";
#include "SampleSegment.h"
#include "MediaChannel.h"
// virtual
SampleSegment::~SampleSegment(void)
{
}
// virtual
bool
SampleSegment::Play(MediaChannel *pMediaChannel,
                    CallToken& ct)
{
        return (bool)(pMediaChannel->DoPlay(ct, cFilename, cIsTTS, false) == MSPF_OK);
}
```

```
/*
 * SCChannel.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.8 $
 * $Date: 2000/12/07 03:19:51 $
 */
static const char* id="$Id: SCChannel.cpp,v 1.8 2000/12/07 03:19:51 Mitsuru Oshima Exp $";
#include "SCChannel.h"
#include "VPPMsg/VPPDefs.h"
// virtual
SCChannel::~SCChannel(void)
{
}
unsigned long
SCChannel::ListenTo(SCChannel *pOtherChannel)
{
        unsigned long ret;

        if (pOtherChannel)
                ret = ListenTo(pOtherChannel->cTxTimeslot);
        else
                ret = ListenTo((unsigned short)-1);
        if (ret == TSPF_OK)
        {
                if (cChannelImListeningTo)
                        cChannelImListeningTo->NotifyChannelNoLongerListening(this);

                cChannelImListeningTo = pOtherChannel;

                if (cChannelImListeningTo)
                        cChannelImListeningTo->NotifyListeningChannel(this);
        }
        return ret;
}
void
SCChannel::NotifyListeningChannel(SCChannel *pListeningChannel)
{
        std::list<SCChannel *>::iterator chiter = cChannelsListeningToMe.begin();

        while (chiter != cChannelsListeningToMe.end())
        {
                SCChannel *pChannel = *chiter;
                if (*chiter == pListeningChannel)
                        return;
                chiter++;
        }
        cChannelsListeningToMe.push_back(pListeningChannel);
}
void
SCChannel::NotifyChannelNoLongerListening(SCChannel *pChannelNoLongerListening)
```

```
{
	std::list<SCChannel *>::iterator chiter = cChannelsListeningToMe.begin();

	while (chiter != cChannelsListeningToMe.end())
	{
		SCChannel *pChannel = *chiter;
		if (*chiter == pChannelNoLongerListening)
		{
			cChannelsListeningToMe.remove(pChannelNoLongerListening);
			return;
		}
		chiter++;
	}
}
```

Document ID: 20020402.1

```
/*
 * SecureHash.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:22:19 $
 */
static const char* id="$Id: SecureHash.cpp,v 1.2 2000/12/07 03:22:19 Mitsuru Oshima Exp $";
#include "SecureHash.h"
gm::string
SecureHash::hash(gm::string &str) {
   long tmp[5] = {0.0,0.0,0.0,0.0,0.0};
   int p = 0;
   for (int i=0 ; i<str.length(); i++) {
      p += str.at(i) * (i+11);
   }
   int d = p;
   for (i=0 ; i<str.length(); i++, d++) {
      d = d % str.length();
      long v = str.at(d);
      v = v << ( (d + 50) % 56);
      tmp[i % 5] += v;
   }
   gm::string b;
   toPrintableString(b, tmp[p++ % 5]);
   toPrintableString(b, tmp[p++ % 5]);
   toPrintableString(b, tmp[p++ % 5]);
   toPrintableString(b, tmp[p++ % 5]);
   toPrintableString(b, tmp[p++ % 5]);
   return b;
}
void
SecureHash::toPrintableString(gm::string &str, unsigned long v) {
   unsigned long mod = 0;
   while (v > 0) {
      mod = v % 62;
      v = (v - mod)/62;
      if (mod < 10) {
         str += (char) ('0' + mod);
      } else if (mod < 36) {
         str += (char) ('a' + (mod - 10));
      } else {
         str += (char) ('A' + (mod - 36));
      }
   }
}
```

```
/*
 * ServiceProvider.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.10 $
 * $Date: 2000/12/07 03:19:51 $
 */
static const char* id="$Id: ServiceProvider.cpp,v 1.10 2000/12/07 03:19:51 Mitsuru Oshima Exp $";
#include "ServiceProvider.h"
#include "Channel.h"
#include "VPPMsg/FailureNtfy.h"
#include "MTSServer.h"
#if 0
#include "MTSServerNode.h"
#endif
#include "VPPMsg/Property.h"
#include "SessionNode.h"
// virtual
ServiceProvider::~ServiceProvider(void)
{
  Shutdown();
}
void
ServiceProvider::Shutdown() {
  if (!shutdown) {
    shutdown = true;
    std::list<Channel *>::iterator iter = cChannels.begin();

    while (iter != cChannels.end())
    {
      delete *iter;
      iter++;
    }
  }
}
// virtual
bool
ServiceProvider::PostSetup(gm::string registryParent)
{
        return true;
}
Channel *
ServiceProvider::FindChannelForCallToken(CallToken& callToken)
{
        if (!callToken.IsValid())
                return 0;
        std::list<Channel *>::iterator iter = cChannels.begin();
        while (iter != cChannels.end() &&
                (*iter)->GetCurrentCallToken() != callToken)
                iter++;
        return (iter != cChannels.end()) ? *iter : 0;
```

```
}
#if 0
Channel *
ServiceProvider::FindChannelForClientID(ClientID& clientID)
{
        std::list<Channel *>::iterator iter = cChannels.begin();
        while (iter != cChannels.end() && (*iter)->GetClientID() != clientID)
                iter++;
        return (iter != cChannels.end()) ? *iter : 0;
}
#endif
void
ServiceProvider::AddChannel(Channel *pChannel)
{
        cChannels.push_back(pChannel);
}
#if 0
CMSMessage::SerialNumberT
ServiceProvider::NotifyFailure(ClientID& VUIClientID, CallToken& ct,
        CMSMessage::SerialNumberT FailureSerialNo, unsigned long Info)
{
        FailureNotification ntfy(ct, cFailureNotificationID, FailureSerialNo, Info);
        MTSServerNode *pServerNode = cServer->FindServerNodeForClientID(VUIClientID);
        if (pServerNode)
                return pServerNode->WriteWrappedMessage(ntfy, VUIClientID);
        else
                return 0;
}
CMSMessage::SerialNumberT
ServiceProvider::NotifyFailure(ClientID& VUIClientID,
        CMSMessage::SerialNumberT FailureSerialNo, unsigned long Info)
{
        CallToken invalid;
        return NotifyFailure(VUIClientID, invalid, FailureSerialNo, Info);
}
#endif
bool
ServiceProvider::SetDefaultProperty(const gm::string& name, const gm::string& value)
{
        return GetMTSServer()->GetSessionMgr()->SetDefaultProperty(name, value);
}
bool
ServiceProvider::SetDefaultProperty(const gm::string& name, long value)
{
        return GetMTSServer()->GetSessionMgr()->SetDefaultProperty(name, value);
}
#if 0
// virtual
bool
ServiceProvider::NotifyCallAboutToBeDropped(MTSServerNode *pServerNode,
        ClientID& VUIClientID, CallToken& ct)
{
        return true;
}
#endif
```

```
/*
 * SessionLogger.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.10 $
 * $Date: 2000/12/07 03:19:51 $
 */
static const char* id="$Id: SessionLogger.cpp,v 1.10 2000/12/07 03:19:51 Mitsuru Oshima Exp $";
#include <windows.h>
#include <time.h>
#include <direct.h>
#include <stdio.h>
#include <io.h>
#include "SessionLogger.h"
#include "SessionNode.h"
#include "SessionMgr.h"
#include "VPPUtils/GMString"
#include "VPPUtils/ConfigMgr.h"
#include "VPPMsg/Property.h"
#if 0
#include "VPPMsg/PhysicalMachineName.h"
#endif
#include "SpeechChannel.h"
#include "TelephonyChannel.h"
char *SessionLogger::cMonths[12] = {
        "January",
        "February",
        "March",
        "April",
        "May",
        "June",
        "July",
        "August",
        "September",
        "October",
        "November",
        "December"
};
SessionLogger::SessionLogger(SessionNode *pSessionNode, TelephonyChannel *pTChannel)
: cLogDirCreated(false)
{
        cFile = 0;
        cInState = false;
        cSessionNode = pSessionNode;
        ConfigMgr::GetString(MTS_REGISTRY_BASE, "DefaultLogRoot", cLogRoot);
        if (pTChannel)
                cIDString = pTChannel->GetIDString();
        CreateLogPath();
}
SessionLogger::~SessionLogger(void)
{
```

```
        CloseFile();
}
bool
SessionLogger::CreateLogPath(void)
{
        if (cLogPath.empty())
                CreateLogPathname();
        return true;
}
bool
SessionLogger::CreateLogDirectory(void)
{
        if (cLogDirCreated)
                return true;
        gm::string copy = cLogPath.c_str();
        char *ptr = (char *)copy.c_str();
        int ndx = 0;
        if (!copy.empty())
        {
                int totallen = strlen(ptr);
                // start from the beginning, stopping at each / or \,
                // creating the directories, if necessary
                //
                if (ptr[ndx] == '\\')
                        ndx = 2;
                while (ndx < totallen)
                {
                        while (ndx < totallen && ptr[ndx] != '/' && ptr[ndx] != '\\')
                                ndx++;
                        ptr[ndx] = 0;
                        if (_access(ptr, 0) == -1)
                        {
                                if (_mkdir(ptr) == -1)
                                        return false;
                        }
                        ptr[ndx] = '\\';
                        ndx++;
                }
        }
        else
        {
                cLogPath = ".";
                return false;
        }
        cLogDirCreated = true;
        return true;
}
static char* GetHostName() {
   static char* hname = NULL;
   if (hname == NULL) {
      hname = new char[ MAX_COMPUTERNAME_LENGTH + 1 ];
         DWORD size = MAX_COMPUTERNAME_LENGTH + 1;
         if (!::GetComputerName(hname, &size))
        strcpy(hname, "localhost");
   }
   return hname;
```

```
}
void
SessionLogger::CreateLogPathname(void)
{
        // PhysicalMachineName ThisPhysicalMachine;
        cLogPath = cLogRoot.c_str();
        if (cLogPath[cLogPath.length() - 1] != '\\' &&
                cLogPath[cLogPath.length() - 1] != '/')
                cLogPath += "\\";
        time_t timet;
        time(&timet);
        struct tm *tmtime = localtime(&timet);
        char buf[20];
        sprintf(buf, "%4d", 1900 + tmtime->tm_year);
        cLogPath += buf;
        cLogPath += "\\";
        sprintf(buf, "%02d", tmtime->tm_mon + 1);
        cLogPath += buf;
        cLogPath += cMonths[tmtime->tm_mon];
        cLogPath += "\\";
        sprintf(buf, "%02d", tmtime->tm_mday);
        cLogPath += buf;
        cLogPath += "\\";
        sprintf(buf, "%02d", tmtime->tm_hour);
        cLogPath += buf;
        cLogPath += '-';
        sprintf(buf, "%02d", tmtime->tm_min);
        cLogPath += buf;
        cLogPath += '-';
        sprintf(buf, "%02d", tmtime->tm_sec);
        cLogPath += buf;
        cLogPath += '-';
        cLogPath += GetHostName();
        cLogPath += '-';
        cLogPath += cIDString;
}
bool
SessionLogger::OpenFile(const gm::string& fullpath)
{
        cLogFile = fullpath.c_str();
        cFile = fopen(fullpath, "wt");
        if (cFile != 0)
        {
                Property prop("Logging.LogfilePath", cLogPath);
                cSessionNode->SetProperty(prop);
                prop.Setup("Logging.Logfile", fullpath);
                cSessionNode->SetProperty(prop);
        }

        return (bool)(cFile != 0);
}
bool
SessionLogger::CloseFile(void)
{
        if (cFile)
        {
```

```
                if (cInState)
                        NotifyLeaveState();
                LogItem(";;;");
                gm::string item;
                item = ";;; Closed ";
                item += cLogFile;
                LogItem(item);
                gm::string nicetime;
                GetNiceTime(nicetime);
                item = ";;; ";
                item += nicetime;
                LogItem(item);
                LogItem(";;;");
                gm::string empty;
                Property prop("Logging.LogfilePath", empty);
                cSessionNode->SetProperty(prop);
                prop.Setup("Logging.Logfile", empty);
                cSessionNode->SetProperty(prop);
                fclose(cFile);
                cFile = 0;
                cSessionNode->cSpeechChannel->NotifyStopLogging();
                return true;
        }
        return false;
}
void
SessionLogger::GetNiceTime(gm::string& str)
{
        time_t timet;
        time(&timet);
        struct tm *tmtime = localtime(&timet);
        str = asctime(tmtime);
        str.erase(str.length() - 1, 1);
}
bool
SessionLogger::NotifyEnteringState(const gm::string& StateName)
{
        if (cInState)
                NotifyLeaveState();
        LogItem(";;;;;;;;;;");
        LogItem("start{");
        cInState = true;
        gm::string nicetime;
        GetNiceTime(nicetime);
        LogItem("TIME", nicetime);
        LogItem("STATE", StateName);
        cStateBeginTime = ::GetTickCount();
        float elapsed = ((cStateBeginTime - cSessionBeginTime) / 1000.0);
        char buf[20];
        sprintf(buf, "%.3f", elapsed);
        return LogItem("BEGIN_TIME", buf);
}
bool
SessionLogger::LogItem(const gm::string& ItemName, const gm::string& ItemValue)
{
        if (cFile)
```

```
        {
                if (cInState)
                        fprintf(cFile, "\t%s = %s\n", ItemName.c_str(),
                                ItemValue.c_str() ? ItemValue.c_str() : "");
                else
                        fprintf(cFile, "%s\n", ItemName.c_str());
                return true;
        }
        return false;
}
bool
SessionLogger::LogItem(const gm::string& ItemName, unsigned short ItemValue)
{
        char buf[20];
        _itoa(ItemValue, buf, 10);
        return LogItem(ItemName, buf);
}
bool
SessionLogger::LogItem(const gm::string& ItemName, float ItemValue)
{
        char buf[50];
        ::sprintf(buf, "%.3f", ItemValue);
        return LogItem(ItemName, buf);
}
bool
SessionLogger::NotifyLeaveState(void)
{
        float elapsed = ((::GetTickCount() - cStateBeginTime) / 1000.0);
        char buf[20];
        sprintf(buf, "%.3f", elapsed);
        LogItem("STATE_DURATION", buf);
        cInState = false;
        return LogItem("}end");
}
bool
SessionLogger::NotifyPropertySet(Property& prop)
{
        if (prop.GetName() == "Logging.Enable")
        {
                if (prop.GetLongValue() != 0)
                {
                        // Logging is being turned on
                        //
                        if (!CreateLogDirectory())
                                return false;
                        if (!cFile)
                        {
                                //PhysicalMachineName ThisPhysicalMachine;
                                gm::string logfile = cLogPath.c_str();
                                logfile += "\\LOG";
                                OpenFile(logfile);
                                LogItem(";;;");
                                gm::string str, item;
                                item = ";;; Opened ";
                                item += logfile;
                                LogItem(item);
```

```
                    GetNiceTime(str);
                    item = ";;; Time: ";
                    item += str;
                    LogItem(item);
                    item = ";;; Host: ";
                    item += GetHostName(); //ThisPhysicalMachine.GetName();
                    LogItem(item);
                    item = ";;; Base path: ";
                    item += cLogRoot;
                    LogItem(item);
                    // LOGGING_TBD - log the ";;; Program: ..." item
                    //
                    // LOGGING_TBD - log the ";;; Package: ..." item
                    //
                    LogItem(";;;");
                    cSessionBeginTime = ::GetTickCount();
                    cSessionNode->cSpeechChannel->NotifyStartLogging();
                }
            }
            else
            {
                // Logging is being turned off
                //
                CloseFile();
            }
            return true;
        }
        else if (prop.GetName() == "Logging.LogRoot")
            cLogRoot = prop.GetValue();
        return false;
    }
```

```
/*
 * SessionMgr.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.53 $
 * $Date: 2000/12/07 03:19:52 $
 */
static const char* id="$Id: SessionMgr.cpp,v 1.53 2000/12/07 03:19:52 Mitsuru Oshima Exp $";
#include <iostream.h>
#include "SessionMgr.h"
#include "TelephonyChannel.h"
#include "MediaChannel.h"
#include "SpeechChannel.h"
#include "TTSChannel.h"
#include "SessionNode.h"
#include "TelephonyServiceProvider.h"
#include "MediaServiceProvider.h"
#include "MTSServer.h"
#include "VPPMsg/Property.h"
#include "SpeechServiceProvider.h"
#include "TTSServiceProvider.h"
#include "PromptSegment.h"
#include "CMS/CMSServerNode.h"
#include "VPPUtils/Synchronized.h"
#include "VPPUtils/ConfigMgr.h"
#include "VPPUtils/Log.h"
#include "ConnectionMgr.h"
#include "VUISessionImpl.h"
SessionMgr *SessionMgr::cSessionMgr = 0;
class SESSLog : public LogT<SESSLog> {
  SESSLog() : LogT<SESSLog>("SESSION ") {};
  static SESSLog log;
};
SESSLog SESSLog::log;
SessionMgr::SessionMgr(MTSServer *pMTSServer, MTSServerImpl* impl) : zombieList(5),
cMTSServer(pMTSServer), cMTSServerImpl(impl)
{
        // Setup some non-service provider related default properties here
        //
        SetDefaultProperty("Logging.Enable", 0);
        SetDefaultProperty("Logging.WriteEndpointed", 1);
        gm::string str(".");
        ConfigMgr::GetString(MTS_REGISTRY_BASE, "DefaultLogRoot", str);
        SetDefaultProperty("Logging.LogRoot", str);
  if (ConfigMgr::GetString(MTS_REGISTRY_BASE ".Media Service Providers", "DefaultAdjustPlayVolumeDB",
str))
                SetDefaultProperty("Media.DefaultAdjustPlayVolumeDB", ::atoi(str.c_str()));
  else
                SetDefaultProperty("Media.DefaultAdjustPlayVolumeDB", 0);
        // set the default setting whether to do WholeCallRecording
        if (ConfigMgr::GetString(MTS_REGISTRY_BASE ".Logging", "EnableWholeCallRecording", str))
```

```
                SetDefaultProperty("Logging.EnableWholeCallRecording", ::atoi(str.c_str()));
        else
                SetDefaultProperty("Logging.EnableWholeCallRecording", 0);
}
SessionMgr::~SessionMgr(void)
{
  {
        Synchronized sync(this);
        std::list<SessionNode *>::iterator iter = cSessions.begin();
        while (iter != cSessions.end())
        {
                delete *iter;
                iter++;
        }
  }
}
SessionNode *
SessionMgr::FindSessionNode(Channel *pCh)
{
        Synchronized sync(this);
        SessionNode *pNode = 0;
        std::list<SessionNode *>::iterator aciter = cSessions.begin();
        while (aciter != cSessions.end() && !pNode)
        {
                SessionNode *pLocalNode = *aciter;
                if (pLocalNode->cTelephonyChannel == pCh ||
                        pLocalNode->cPlayMediaChannel == pCh ||
                        pLocalNode->cRecordMediaChannel == pCh ||
                        pLocalNode->cSpeechChannel == pCh ||
                        pLocalNode->cTTSChannel == pCh)
                        pNode = pLocalNode;
                aciter++;
        }
        return pNode;
}
SessionNode *
SessionMgr::FindSessionNode(TelephonyChannel *pCh)
{
        return FindSessionNode((Channel *)pCh);
}
SessionNode *
SessionMgr::FindSessionNode(MediaChannel *pCh)
{
        return FindSessionNode((Channel *)pCh);
}
SessionNode *
SessionMgr::FindSessionNode(SpeechChannel *pCh)
{
        return FindSessionNode((Channel *)pCh);
}
SessionNode *
SessionMgr::FindSessionNode(TTSChannel *pCh)
{
        return FindSessionNode((Channel *)pCh);
}
#if 0
```

```
SessionNode *
SessionMgr::FindSessionNode(ClientID& VUIClientID)
{
	Synchronize sync(cCriticalSection);
	SessionNode *pNode = 0;
	std::list<SessionNode *>::iterator aciter = cSessions.begin();
	while (aciter != cSessions.end() && !pNode)
	{
		if ((*aciter)->cClientID == VUIClientID)
			pNode = *aciter;
		aciter++;
	}
	return pNode;
}
SessionNode *
SessionMgr::FindSessionNode(ServerID& VUIBrokerServerID)
{
	Synchronize sync(cCriticalSection);
	SessionNode *pNode = 0;
	std::list<SessionNode *>::iterator aciter = cSessions.begin();
	while (aciter != cSessions.end() && !pNode)
	{
		if ((*aciter)->cClientID.GetServerID() == VUIBrokerServerID)
			pNode = *aciter;
		aciter++;
	}
	return pNode;
}
#endif
SessionNode *
SessionMgr::FindSessionNode(CallToken& callToken)
{
	Synchronized sync(this);
	SessionNode *pNode = 0;
	std::list<SessionNode *>::iterator aciter = cSessions.begin();
	while (aciter != cSessions.end() && !pNode)
	{
		SessionNode *pThisNode = (SessionNode *)(*aciter);
		if (pThisNode->cTelephonyChannel->GetCurrentCallToken() == callToken)
			pNode = pThisNode;
		aciter++;
	}
	return pNode;
}
SessionNode *
SessionMgr::FindSessionNode(RequestToken& requestToken)
{
	Synchronized sync(this);
	SessionNode *pNode = 0;
	std::list<SessionNode *>::iterator aciter = cSessions.begin();
	while (aciter != cSessions.end() && !pNode)
	{
		SessionNode *pThisNode = (SessionNode *)(*aciter);
		if (pThisNode->cTelephonyChannel->GetCurrentRequestToken() == requestToken)
			pNode = pThisNode;
		aciter++;
```

```
        }
        return pNode;
}
#if 0
SessionNode *
SessionMgr::FindSessionNode(ClientID& VUIClientID, CallToken& callToken)
{
        Synchronize sync(cCriticalSection);
        SessionNode *pNode = 0;
        std::list<SessionNode *>::iterator aciter = cSessions.begin();
        while (aciter != cSessions.end() && !pNode)
        {
                SessionNode *pThisNode = (SessionNode *)(*aciter);
                if (pThisNode->cClientID == VUIClientID &&
                        callToken == pThisNode->cTelephonyChannel->GetCurrentCallToken())
                        pNode = pThisNode;
                aciter++;
        }
        return pNode;
}
SessionNode *
SessionMgr::FindSessionNode(ClientID& VUIClientID, RequestToken& requestToken)
{
        Synchronize sync(cCriticalSection);
        SessionNode *pNode = 0;
        std::list<SessionNode *>::iterator aciter = cSessions.begin();
        while (aciter != cSessions.end() && !pNode)
        {
                SessionNode *pThisNode = (SessionNode *)(*aciter);
                if (pThisNode->cClientID == VUIClientID &&
                        requestToken == pThisNode->cTelephonyChannel->GetCurrentRequestToken())
                        pNode = pThisNode;
                aciter++;
        }
        return pNode;
}
#endif
void
SessionMgr::Initialize(void)
{
        std::list<Channel *>& tchannels = cMTSServer->GetTelephonyServiceProvider()->GetChannels();
        std::list<Channel *>::iterator tchiter = tchannels.begin();
        std::list<Channel *>& mchannels = cMTSServer->GetMediaServiceProvider()->GetChannels();
        std::list<Channel *>::iterator mchiter = mchannels.begin();
   maxZombieSize = cMTSServer->GetTelephonyServiceProvider()->GetNumChannels();
        // Configure the media and telephony channels
        //
        if (!cMTSServer->GetTelephonyServiceProvider()->IncomingRequiresMediaChannel())
        {
                while (tchiter != tchannels.end())
                {
                        ((TelephonyChannel *)(*tchiter))->ListenTo((SCChannel *)0);
                        tchiter++;
                }
                while (mchiter != mchannels.end())
                {
```

```
                ((MediaChannel *)(*mchiter))->ListenTo((SCChannel *)0);
                mchiter++;
            }
        }
        else
        {
            while (tchiter != tchannels.end())
            {
                if (((TelephonyChannel *)(*tchiter))->GetCapabilities() &
                    TelephonyChannel::C_Incoming)
                {
                    ((MediaChannel *)(*mchiter))->ListenTo((SCChannel *)(*tchiter));
                    mchiter++;
                }
                tchiter++;
            }
            while (mchiter != mchannels.end())
            {
                ((MediaChannel *)(*mchiter))->ListenTo((SCChannel *)0);
                mchiter++;
            }
        }
}
bool
SessionMgr::CreateTrombone(TelephonyChannel *pCh1, TelephonyChannel *pCh2, bool FullDuplex)
{
#ifdef OLD
        SessionNode *pNode1 = FindSessionNode(pCh1);
        SessionNode *pNode2 = FindSessionNode(pCh2);
        if (pNode1)
            pNode1->cTelephonyChannel->ListenTo(pNode2->cTelephonyChannel);

        if (FullDuplex && pNode2)
            pNode2->cTelephonyChannel->ListenTo(pNode1->cTelephonyChannel);

        return pNode1 || pNode2;
#else
        if (pCh1)
            pCh1->ListenTo(pCh2);
        if (FullDuplex && pCh2)
            pCh2->ListenTo(pCh1);
        return true;
#endif
}
bool
SessionMgr::ResetRouting(TelephonyChannel *pCh)
{
        SessionNode *pNode = FindSessionNode(pCh);
        if (pNode && pNode->cPlayMediaChannel)
            pNode->cTelephonyChannel->ListenTo(pNode->cPlayMediaChannel);
        return pNode != 0;
}
bool
SessionMgr::BreakTrombone(TelephonyChannel *pCh1, TelephonyChannel *pCh2)
{
        if (pCh1)
```

```
                ResetRouting(pCh1);
        if (pCh2 && pCh2 != pCh1)
                ResetRouting(pCh2);
        return pCh1 || pCh2;
}
#if 0
TelephonyChannel *
SessionMgr::GetTelephonyChannelFor(ClientID& VUIClientID)
{
        Synchronize sync(cCriticalSection);
        SessionNode *pNode = FindSessionNode(VUIClientID);
        return pNode ? pNode->cTelephonyChannel : 0;
}
TelephonyChannel *
SessionMgr::GetTelephonyChannelFor(ServerID& VUIBrokerServerID)
{
        Synchronize sync(cCriticalSection);
        SessionNode *pNode = FindSessionNode(VUIBrokerServerID);
        return pNode ? pNode->cTelephonyChannel : 0;
}
#endif
TelephonyChannel *
SessionMgr::GetFirstTelephonyChannel(void)
{
        Synchronized sync(this);
        std::list<SessionNode *>::iterator iter = cSessions.begin();
        if (iter != cSessions.end())
                return (*iter)->cTelephonyChannel;
        return 0;
}
VUISessionImpl *
SessionMgr::CreateSession(TelephonyChannel *pTCh, Address& Target, Address& CallingNumber)
{
  bool channelsForWholeCallRecordingAllocated = false;
  VUISessionImpl *pNode = NULL;
  {
    Synchronized sync(this);
        pNode = new VUISessionImpl(this, pTCh);
        pTCh->NotifyBoundToSession(pNode);
        if (pTCh->GetCurrentCallToken().IsValid())
                pTCh->NotifyBoundToCall(pTCh->GetCurrentCallToken());
        // Save Called and Calling numbers for CDR_LOG
        pNode->cTarget = Target;
        pNode->cCallingNumber = CallingNumber;
        // Find two unbound media channels, an unbound speech channel, and an
        // unbound TTS channel to serve this session
        //
        // If this telephony channel already has a listener, this must be an incoming
        // channel on a TSP that requires a media listening for incoming calls.  In that
        // case, use that existing media channel for recording on this telephony channel.
        // Otherwise, just find an idle one, as usual
        //
        std::list<SCChannel *>& listeners = pTCh->GetChannelsListeningToMe();
        std::list<SCChannel *>::iterator listenersiter = listeners.begin();
        if (listenersiter != listeners.end())
        {
```

```
                                        }
                                }
                        }
                        mchiter++;
                }
                if (cMTSServer->GetSpeechServiceProvider())
                {
                        Synchronized sync(this);
                        while (!(pNode->cSpeechChannel = cMTSServer->GetSpeechServiceProvider()-
>FindAvailableChannel()))
                        {
                                sync.wait();
                        }
                }
                pNode->cSpeechChannel->NotifyBoundToSession(pNode);
                if (pTCh->GetCurrentCallToken().IsValid())
                        pNode->cSpeechChannel->NotifyBoundToCall(pTCh->GetCurrentCallToken());
                if (cMTSServer->GetTTSServiceProvider())
                {
                                Synchronized sync(this);
        while (!(pNode->cTTSChannel = cMTSServer->GetTTSServiceProvider()->FindAvailableChannel()))
                        {
                                sync.wait();
                        }
        }
                        pNode->cTTSChannel->NotifyBoundToSession(pNode);
                        if (pTCh->GetCurrentCallToken().IsValid())
                                pNode->cTTSChannel->NotifyBoundToCall(pTCh->GetCurrentCallToken());
                // Make the record media channel listen to the telephony channel
                //
                if (pNode->cRecordMediaChannel)
                {
                        pNode->cRecordMediaChannel->ListenTo(pNode->cTelephonyChannel);
                }
                // Make the telephony channel listen to the play media channel
                //
                if (pNode->cPlayMediaChannel)
                {
                        pNode->cTelephonyChannel->ListenTo(pNode->cPlayMediaChannel);
                        if (channelsForWholeCallRecordingAllocated)
                        {
                                pNode->cOutboundRecordMediaChannel->ListenTo(pNode->cPlayMediaChannel);
                        }
                }
                if (channelsForWholeCallRecordingAllocated)
                {
                        pNode->cInboundRecordMediaChannel->ListenTo(pNode->cTelephonyChannel);
                }
                cSessions.push_back(pNode);
        }

        PropertyList props;
        pNode->SetProperties(&props);
        // no need to sync on SessionMgr
                if (!channelsForWholeCallRecordingAllocated)
                {
```

```
                pNode->cInboundRecordMediaChannel = 0;
                pNode->cOutboundRecordMediaChannel = 0;
                if (pNode->DoWholeCallRecording())
                        SESSLog::warning("Cannot allocate channels to do whole call recording.");
        }
        else
        {
                if (pNode->cSessionLogger.CreateLogDirectory())
                {
                        gm::string sessionLogFullPath = pNode->cSessionLogger.GetLogPath();
                        gm::string recordFullPath = sessionLogFullPath;

                        recordFullPath += "\\wholecallincoming.wav";
                        pNode->cInboundRecordMediaChannel->Record(recordFullPath, false, -1, -1, -1);
                        recordFullPath = sessionLogFullPath;
                        recordFullPath += "\\wholecallprompt.wav";
                        pNode->cOutboundRecordMediaChannel->Record(recordFullPath, false, -1, -1, -1);
                }
        }
        SESSLog::diag("Created Node [%ld] with TChannel %d, PlayMChannel %d, RecordMChannel %d,
SpeechChannel %d, OutRecordChannel %d, InRecordChannel %d.",
                pNode->GetID(),
                pNode->cTelephonyChannel ? pNode->cTelephonyChannel->GetID() : -1,
                pNode->cPlayMediaChannel ? pNode->cPlayMediaChannel->GetID() : -1,
                pNode->cRecordMediaChannel ? pNode->cRecordMediaChannel->GetID() : -1,
                pNode->cSpeechChannel ? pNode->cSpeechChannel->GetID() : -1,
                pNode->cOutboundRecordMediaChannel ? pNode->cOutboundRecordMediaChannel->GetID() : -1,
                pNode->cInboundRecordMediaChannel ? pNode->cInboundRecordMediaChannel->GetID() : -1);

        return pNode;
}
bool
SessionMgr::DestroySession(TelephonyChannel *pTCh)
{
        Synchronized sync(this);
        VUISessionImpl *pNode = static_cast<VUISessionImpl*>(FindSessionNode(pTCh));
        if (pNode)
        {
    pNode->diag("destroying session");
    Synchronized sync(pNode);
                // SessionLock session(pNode);
                // definitely no need to check for session->IsGoingAway()
                pNode->cTelephonyChannel->ListenTo((SCChannel *)0);
                pNode->cTelephonyChannel->NotifyBoundToSession(0);
                if (pNode->cPlayMediaChannel)
                {
                        pNode->cPlayMediaChannel->ListenTo((SCChannel *)0);
                        pNode->cPlayMediaChannel->NotifyBoundToSession(0);
                }
                if (pNode->cRecordMediaChannel)
                {
                        if (!cMTSServer->GetTelephonyServiceProvider()->IncomingRequiresMediaChannel())
                                pNode->cRecordMediaChannel->ListenTo((SCChannel *)0);
                        pNode->cRecordMediaChannel->NotifyBoundToSession(0);
                }
                if (pNode->cOutboundRecordMediaChannel)
```

```
                {
                        pNode->cOutboundRecordMediaChannel->Stop();
                        pNode->cOutboundRecordMediaChannel->ListenTo((SCChannel *)0);
                        pNode->cOutboundRecordMediaChannel->NotifyBoundToSession(0);
                }
                if (pNode->cInboundRecordMediaChannel)
                {
                        pNode->cInboundRecordMediaChannel->Stop();
                        pNode->cInboundRecordMediaChannel->ListenTo((SCChannel *)0);
                        pNode->cInboundRecordMediaChannel->NotifyBoundToSession(0);
                }
                if (pNode->cSpeechChannel)
                        pNode->cSpeechChannel->NotifyBoundToSession(0);
                if (pNode->cTTSChannel)
                        pNode->cTTSChannel->NotifyBoundToSession(0);
        pNode->cSessionLogger.CloseFile();
                cSessions.remove(pNode);
        try {
          pNode->deactivate();
        } catch (CORBA::Exception &e) {
          pNode->diag("could not deactivate the session. Probaby done by DGC");
          cerr << "Could not deactivate the session" << e << endl;
        }
                //session.ReleaseNode();
                ConnectionMgr::removeConnectionListener(pNode);
        moveToZombieList(pNode);
                // delete pNode;
                return true;
            }
            return false;
}
void
SessionMgr::moveToZombieList(SessionNode* node) {
  Synchronized sync(this);
  if (zombieList.contains(node)) {
    // already deleted?
    SESSLog::warning("SessionNode(%d) is already in zombie list.",
                node->GetID());
    return;
  }
  zombieList.push(node);
  while (zombieList.size() > maxZombieSize) {
    SessionNode* node = zombieList.pop();
    SESSLog::diag("deleting session node (%d).", node->GetID());
    delete node;
  }
}
bool
SessionMgr::SetDefaultProperty(const gm::string& name, const gm::string& value)
{
        Synchronized sync(this);
        return cDefaultProperties.SetProperty(name, value);
}
bool
SessionMgr::SetDefaultProperty(const gm::string& name, long value)
{
```

```
        Synchronized sync(this);
        return cDefaultProperties.SetProperty(name, value);
}
bool
SessionMgr::GetDefaultProperty(const gm::string& name, gm::string& value)
{
        Synchronized sync(this);
        return cDefaultProperties.GetProperty(name, value);
}
bool
SessionMgr::GetDefaultProperty(const gm::string& name, long& value)
{
        Synchronized sync(this);
        return cDefaultProperties.GetProperty(name, value);
}
#if 0
MTSServerNode*
SessionMgr::FindServerNodeForClientID(ClientID& id) {
  return cMTSServer->FindServerNodeForClientID(id);
}
#endif
void
SessionMgr::SetSessionMgr(SessionMgr* s) {
  cSessionMgr = s;
}
SessionMgr*
SessionMgr::GetSessionMgr() {
  return cSessionMgr;
}
```

```
/*
 * SessionNode.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.32 $
 * $Date: 2000/12/07 03:19:52 $
 */
static const char* id="$Id: SessionNode.cpp,v 1.32 2000/12/07 03:19:52 Mitsuru Oshima Exp $";
#include <Windows.h>
#include "SessionNode.h"
#include "SessionMgr.h"
#include "TelephonyChannel.h"
#include "MediaChannel.h"
#include "SpeechChannel.h"
#include "TTSChannel.h"
#include "MTSServer.h"
#include "TelephonyServiceProvider.h"
#include "MediaServiceProvider.h"
#include "CallTokenMgr.h"
#include "VPPUtils/Synchronized.h"
// message
// message
#include "VPPMsg/DropCallMsg.h"
#include "VPPMsg/PlaceCallOnHoldMsg.h"
#include "VPPMsg/AcceptCallMsg.h"
#include "VPPMsg/RejectCallMsg.h"
#include "VPPMsg/CancelRequestMsg.h"
#include "VPPMsg/PlayMsg.h"
#include "VPPMsg/StopPlayMsg.h"
#include "VPPMsg/RecordMsg.h"
#include "VPPMsg/StopRecordMsg.h"
#include "VPPMsg/RemoveRecordingMsg.h"
#include "VPPMsg/TTSMsg.h"
#include "VPPMsg/CreateCompositePromptMsg.h"
#include "VPPMsg/AppendCompositePromptPromptMsg.h"
#include "VPPMsg/AppendCompositePromptSilenceMsg.h"
#include "VPPMsg/AppendCompositePromptDTMFMsg.h"
#include "VPPMsg/AppendCompositePromptTTSMsg.h"
#include "VPPMsg/EndCompositePromptMsg.h"
#include "VPPMsg/PlayCompositePromptMsg.h"
#include "VPPMsg/HandoffRecordingMsg.h"
#include "VPPMsg/StartRecognitionMsg.h"
#include "VPPMsg/StopRecognitionMsg.h"
#include "VPPMsg/RecognizeStringMsg.h"
#include "VPPMsg/CreateDynamicGrammarMsg.h"
#include "VPPMsg/DeleteDynamicGrammarMsg.h"
#include "VPPMsg/AddPhraseToDynamicGrammarMsg.h"
#include "VPPMsg/RemovePhraseFromDynamicGrammarMsg.h"
#include "VPPMsg/QueryDynamicGrammarContentsMsg.h"
#include "VPPMsg/InsertDynamicGrammarMsg.h"
// notifications
```

```
#include "VPPMsg/CallProgressNtfy.h"
#include "VPPMsg/CallDroppedNtfy.h"
#include "VPPMsg/RequestCancelledNtfy.h"
#include "VPPMsg/CallConnectedNtfy.h"
#include "VPPMsg/SITDetectedNtfy.h"
#include "VPPMsg/CNGDetectedNtfy.h"
#include "VPPMsg/CallAcceptedNtfy.h"
#include "VPPMsg/CallRejectedNtfy.h"
#include "VPPMsg/PlayingNtfy.h"
#include "VPPMsg/PlayDoneNtfy.h"
#include "VPPMsg/StoppedPlayingNtfy.h"
#include "VPPMsg/RecordingNtfy.h"
#include "VPPMsg/StoppedRecordingNtfy.h"
#include "VPPMsg/TTSPlayingNtfy.h"
#include "VPPMsg/TTSDoneNtfy.h"
#include "VPPMsg/CompositePromptCreatedNtfy.h"
#include "VPPMsg/CompositePromptSegmentAppendedNtfy.h"
#include "VPPMsg/CompositePromptEndedNtfy.h"
#include "VPPMsg/CompositePromptPlayingNtfy.h"
#include "VPPMsg/CompositePromptSegmentPlayingNtfy.h"
#include "VPPMsg/CompositePromptSegmentPlayDoneNtfy.h"
#include "VPPMsg/CompositePromptPlayDoneNtfy.h"
#include "VPPMsg/FailureNtfy.h"
#include "VPPMsg/DTMFNtfy.h"
#include "VPPMsg/RecognitionDoneNtfy.h"
#include "VPPMsg/RecognitionResults.h"
#ifdef NOTIFY_ALL
#include "VPPMsg/BeginSpeechNtfy.h"
#include "VPPMsg/EndSpeechNtfy.h"
#endif
#include "VPPMsg/RecognitionStartedNtfy.h"
#include "VPPMsg/CreateDynamicGrammarResultNtfy.h"
#include "VPPMsg/DeleteDynamicGrammarResultNtfy.h"
#include "VPPMsg/AddPhraseToDynamicGrammarResultNtfy.h"
#include "VPPMsg/RemovePhraseFromDynamicGrammarResultNtfy.h"
#include "VPPMsg/QueryDynamicGrammarContentsResultNtfy.h"
#include "VPPMsg/InsertDynamicGrammarResultNtfy.h"
SessionNode::SessionNode(SessionMgr *pSessionMgr, TelephonyChannel *pTChannel)
  : IDLog("SESSION "), cSessionMgr(pSessionMgr), cSessionLogger(this, pTChannel),
    cGoingAway(false), cDoWholeCallRecording(false)
{
  cTelephonyChannel = pTChannel;
  cPlayMediaChannel = 0;
  cRecordMediaChannel = 0;
  cOutboundRecordMediaChannel = 0;
  cInboundRecordMediaChannel = 0;
  cSpeechChannel = 0;
  cTTSChannel = 0;
  CallToken &ct = pTChannel->GetCurrentCallToken();
  cID = ct.GetSequenceNumber();
  setLogID("[%ld]", cID);

  if (pSessionMgr)
  {
    long value;
    if (pSessionMgr->GetDefaultProperty("Logging.EnableWholeCallRecording", value))
```

```
        cDoWholeCallRecording = value ? true : false;
    }

    //cMutex = (unsigned long)::CreateMutex(NULL, FALSE, NULL);
    //cGoingAwayEvt = (unsigned long)::CreateEvent(NULL, true, false, NULL);
}
SessionNode::~SessionNode(void)
{
}
TelephonyServiceProvider*
SessionNode::GetTSP() {
    return cSessionMgr->GetMTSServer()->GetTelephonyServiceProvider();
}
MediaServiceProvider*
SessionNode::GetMSP() {
    return cSessionMgr->GetMTSServer()->GetMediaServiceProvider();
}
unsigned long
SessionNode::GetID() {
    return cID;
}
bool
SessionNode::SetProperties(PropertyList *pProperties)
{
    Synchronized sync(this);
    if (IsGoingAway())
                return false;
    cProperties = cSessionMgr->GetDefaultProperties();
        cProperties.Merge(*pProperties);
        NotifyPropertiesSet();
        return true;
}
bool
SessionNode::SetProperty(Property& prop)
{
    Synchronized sync(this);
    cProperties.SetProperty(prop.GetName(), prop.GetValue());
    NotifyPropertySet(prop);
    return true;
}
bool
SessionNode::GetProperty(const gm::string& name, gm::string& value)
{
    Synchronized sync(this);
    return cProperties.GetProperty(name, value);
}
bool
SessionNode::GetProperty(const gm::string& name, long& value)
{
    Synchronized sync(this);
    return cProperties.GetProperty(name, value);
}
bool
SessionNode::UpdateCallToken(void)
{
    // synchronized?
```

```
    if (cTelephonyChannel->GetCurrentCallToken().IsValid())
    {
      CallToken& ct = cTelephonyChannel->GetCurrentCallToken();
      if (cPlayMediaChannel)
        cPlayMediaChannel->NotifyBoundToCall(ct);
      if (cRecordMediaChannel)
        cRecordMediaChannel->NotifyBoundToCall(ct);

      if (cSpeechChannel)
        cSpeechChannel->NotifyBoundToCall(ct);

      if (cTTSChannel)
        cTTSChannel->NotifyBoundToCall(ct);

      return true;
    }
    return false;
}
void
SessionNode::NotifyPropertiesSet(void)
{
    std::list<Property *> const &list = cProperties.GetList();
    std::list<Property *>::const_iterator iter = list.begin();

    while (iter != list.end())
    {
      NotifyPropertySet(*(*iter));
      iter++;
    }
}
void
SessionNode::NotifyPropertySet(Property& prop)
{
    if (cTelephonyChannel)
      cTelephonyChannel->NotifyPropertySet(prop);
    if (cPlayMediaChannel)
      cPlayMediaChannel->NotifyPropertySet(prop);
    if (cRecordMediaChannel)
      cRecordMediaChannel->NotifyPropertySet(prop);
    if (cSpeechChannel)
      cSpeechChannel->NotifyPropertySet(prop);
    if (cTTSChannel)
      cTTSChannel->NotifyPropertySet(prop);
    cSessionLogger.NotifyPropertySet(prop);
}
/************************************************/
//
// Media methods
//
/************************************************/
bool
SessionNode::HandleCancelRequest(CancelRequestMessage *pMsg, void *userdata)
{
    Synchronized sync(this);
    if (!IsGoingAway())
    {
```

```
        diag("Cancelling call");
        bool ret = cTelephonyChannel->HandleCancelRequest(this,
                                        pMsg, userdata);
        if (ret)
        {
          cTelephonyChannel->ResetRouting();
          cSessionMgr->DestroySession(cTelephonyChannel);
          diag("Call Cancelled");
          return true;
        }
      } else {
        warning("HandleCancelRequest : session is already terminated");
      }
      return false;
}
// virtual
bool
SessionNode::HandleDropCall(DropCallMessage *pMsg, void *userdata)
{
      Synchronized sync(this);
      if (!IsGoingAway())
      {
        cCDRInfo = pMsg->GetCDRInfo();
        return cTelephonyChannel->HandleDropCall(this, pMsg, userdata);
      }
      else
        NotifyTSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), TSPF_INVALIDCALLTOKEN);
      return true;
}
// virtual
bool
SessionNode::HandlePlaceCallOnHold(PlaceCallOnHoldMessage *pMsg, void *userdata)
{
      // DFB_TBD
      //
      return false;
}
// virtual
bool
SessionNode::HandleAcceptCall(AcceptCallMessage *pMsg, void *userdata)
{
      bool ret = false;
      Synchronized sync(this);
      // cClientID = VUIClientID;
      if (!IsGoingAway())
      {
        ret = cTelephonyChannel->HandleAcceptCall(this, pMsg, userdata);
        // DFB_TBD - only latch the properties if this call is the only one in the session
        // (ie- don't latch if this call was added to an existing session- the properties
        // are already latched in an existing session)
        //
        if (ret)
        {
          // do we really need this? SetProperties(pMsg->GetProperties());
          cCDRInfo = pMsg->GetCDRInfo();
          // the following is commented out because incoming call session
```

```
            // does not start until call gets connected (see ::NotifyCallConnected())
            // CDR_LOG - this is a billable event (inbound call accepted).
            //session->cCDRLogClient.StartCall(&session->cCDRInfo, &session->cTarget,
            //&session->cCallingNumber);
          }
        }
        else
          NotifyTSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), TSPF_INVALIDCALLTOKEN);
        return ret;
      }
      // virtual
      bool
      SessionNode::HandleRejectCall(RejectCallMessage *pMsg, void *userdata)
      {
        bool ret = false;
        Synchronized sync(this);
        // cClientID = VUIClientID;
        if (!IsGoingAway())
        {
          diag("RejectingCall");
          ret = cTelephonyChannel->HandleRejectCall(this, pMsg, userdata);
          if (ret)
          {
            cTelephonyChannel->ResetRouting();
            cSessionMgr->DestroySession(cTelephonyChannel);
            // session.ReleaseNode();
            info("Call Rejected");
          }
        }
        else {
          warning("HandleRejectCall : session is already terminated");
          NotifyTSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), TSPF_INVALIDCALLTOKEN);
        }
        return ret;
      }
      // virtual
      bool
      SessionNode::HandleNotifyCDRInfo(CDRInfoNotification *pMsg, void *userdata)
      {
            bool ret = false;
        Synchronized sync(this);
        if (!IsGoingAway())
          cCDRInfo = pMsg->GetCDRInfo();
        else
          NotifyTSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), TSPF_INVALIDCALLTOKEN);
        return ret;
      }
      /************************************************/
      //
      // Media methods
      //
      /************************************************/
      // virtual
      bool
      SessionNode::HandlePlay(PlayMessage *pMsg, void *userdata)
      {
```

```
    Synchronized sync(this);
    if (! IsGoingAway())
    {
      MediaChannel* play = cPlayMediaChannel;
      MediaChannel* record = cRecordMediaChannel;
      if (play == NULL || record == NULL)
        NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
                MSPF_INVALIDCALLTOKEN);
      else
        return play->HandlePlay(this, pMsg,
                record,
                userdata);
    }
    else
      NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), MSPF_INVALIDCALLTOKEN);
    return true;
}
// virtual
bool
SessionNode::HandleStopPlay(StopPlayMessage *pMsg, void *userdata)
{
    Synchronized sync(this);
    if (!IsGoingAway())
    {
      MediaChannel* play = cPlayMediaChannel;

      if (play == NULL)
        NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
                MSPF_INVALIDCALLTOKEN);
      else
        return play->HandleStop(this, pMsg, userdata);
    }
    else
      NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), MSPF_INVALIDCALLTOKEN);
    return true;
}
// virtual
bool
SessionNode::HandleRecord(RecordMessage *pMsg, void *userdata)
{
    Synchronized sync(this);
    if (!IsGoingAway())
    {
      MediaChannel* record = cRecordMediaChannel;

      if (record == NULL)
        NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
                MSPF_INVALIDCALLTOKEN);
      else
        return record->HandleRecord(this, pMsg, userdata);
    }
    else
      NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), MSPF_INVALIDCALLTOKEN);

    return true;
}
```

```
// virtual
bool
SessionNode::HandleStopRecord(StopRecordMessage *pMsg, void *userdata)
{
  Synchronized sync(this);
  if (!IsGoingAway())
  {
    MediaChannel* record = cRecordMediaChannel;
    if (record == NULL)
      NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
          MSPF_INVALIDCALLTOKEN);
    else
      return record->HandleStop(this, pMsg, userdata);
  }
  else
    NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), MSPF_INVALIDCALLTOKEN);
  return true;
}
// virtual
bool
SessionNode::HandleRemoveRecording(RemoveRecordingMessage *pMsg, void *userdata)
{
  Synchronized sync(this);
  if (!IsGoingAway())
  {
    MediaChannel* record = cRecordMediaChannel;
    if (record == NULL)
      NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
          MSPF_INVALIDCALLTOKEN);
    else
      return record->HandleRemoveRecording(this, pMsg, userdata);
  }
  else
    NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), MSPF_INVALIDCALLTOKEN);

  return true;
}
// virtual
bool
SessionNode::HandleTTS(TTSMessage *pMsg, void *userdata)
{
  Synchronized sync(this);
  if (!IsGoingAway())
  {
    MediaChannel* play = cPlayMediaChannel;
    if (play == NULL)
      NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
          MSPF_INVALIDCALLTOKEN);
    else
      return play->HandleTTS(this, pMsg, userdata);
  }
  else
    NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), MSPF_INVALIDCALLTOKEN);
  return true;
}
// virtual
```

```
bool
SessionNode::HandleCreateCompositePrompt(CreateCompositePromptMessage *pMsg, void *userdata)
{
  Synchronized sync(this);
  if (!IsGoingAway())
  {
    MediaChannel* play = cPlayMediaChannel;
    if (play == NULL)
      NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
          MSPF_INVALIDCALLTOKEN);
    else
      return play->HandleCreateCompositePrompt(this,
                  pMsg, userdata);
  }
  else
    NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), MSPF_INVALIDCALLTOKEN);
  return true;
}
// virtual
bool
SessionNode::HandleAppendCompositePromptPrompt(AppendCompositePromptPromptMessage *pMsg, void
*userdata)
{
  Synchronized sync(this);
  if (!IsGoingAway())
  {
    MediaChannel* play = cPlayMediaChannel;
    if (play == NULL)
      NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
          MSPF_INVALIDCALLTOKEN);
    else
      return play->HandleAppendCompositePromptPrompt(this, pMsg, userdata);
  }
  else
    NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), MSPF_INVALIDCALLTOKEN);
  return true;
}
// virtual
bool
SessionNode::HandleAppendCompositePromptSilence(AppendCompositePromptSilenceMessage *pMsg, void
*userdata)
{
  Synchronized sync(this);
  if (!IsGoingAway())
  {
    MediaChannel* play = cPlayMediaChannel;
    if (play == NULL)
      NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
          MSPF_INVALIDCALLTOKEN);
    else
      return play->HandleAppendCompositePromptSilence(this, pMsg, userdata);
  }
  else
    NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), MSPF_INVALIDCALLTOKEN);
  return true;
}
```

```
bool
SessionNode::HandleAppendCompositePromptDTMF(AppendCompositePromptDTMFMessage *pMsg, void
*userdata)
{
  Synchronized sync(this);
  if (!IsGoingAway())
  {
    MediaChannel* play = cPlayMediaChannel;
    if (play == NULL)
      NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
          MSPF_INVALIDCALLTOKEN);
    else
      return play->HandleAppendCompositePromptDTMF(this, pMsg, userdata);
  }
  else
    NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), MSPF_INVALIDCALLTOKEN);
  return true;
}
// virtual
bool
SessionNode::HandleAppendCompositePromptTTS(AppendCompositePromptTTSMessage *pMsg, void *userdata)
{
  Synchronized sync(this);
  if (!IsGoingAway())
  {
    MediaChannel* play = cPlayMediaChannel;
    if (play == NULL)
      NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
          MSPF_INVALIDCALLTOKEN);
    else
      return play->HandleAppendCompositePromptTTS(this, pMsg, userdata);
  }
  else
    NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), MSPF_INVALIDCALLTOKEN);
  return true;
}
// virtual
bool
SessionNode::HandleEndCompositePrompt(EndCompositePromptMessage *pMsg, void *userdata)
{
  Synchronized sync(this);
  if (!IsGoingAway())
  {
    MediaChannel* play = cPlayMediaChannel;
    if (play == NULL)
      NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
          MSPF_INVALIDCALLTOKEN);
    else
      return play->HandleEndCompositePrompt(this, pMsg, userdata);
  }
  else
    NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), MSPF_INVALIDCALLTOKEN);
  return true;
}
// virtual
bool
```

```
SessionNode::HandlePlayCompositePrompt(PlayCompositePromptMessage *pMsg, void *userdata)
{
  Synchronized sync(this);
  if (!IsGoingAway())
  {
    MediaChannel* play = cPlayMediaChannel;
    if (play == NULL)
      NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
                MSPF_INVALIDCALLTOKEN);
    else
      return play->HandlePlayCompositePrompt(this, pMsg, userdata);
  }
  else
    NotifyMSPFailure(pMsg->GetCallToken(),
              pMsg->GetSerialNo(), MSPF_INVALIDCALLTOKEN);
  return true;
}
bool
SessionNode::GetRecordingFilename(CallToken& ct,
                    RECORDINGID RecordingID, gm::string& filename)
{
        return cRecordMediaChannel->GetRecordingFilename(RecordingID, filename);
}
bool
SessionNode::HandOffRecording(HandOffRecordingMessage *pMsg, void *userdata)
{
  gm::string filename;
  if (GetRecordingFilename(pMsg->GetCallToken(), pMsg->GetRecordingID(),
                filename))
  {
    if (GetMSP()->HandOffRecording(pMsg->GetServiceName(),
                    filename, pMsg->GetData(),
                    pMsg->GetDataLength()))
    {
      return HandleReleaseOwnershipOfRecording(pMsg->GetCallToken(), pMsg->GetRecordingID());
    }
    else
    {
      NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
                MSPF_CANTHANDOFFRECORDING);
      return false;
    }
  }

  NotifyMSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), MSPF_NOSUCHRECORDING);
  return false;
}
// virtual
bool
SessionNode::HandleReleaseOwnershipOfRecording(CallToken& ct, RECORDINGID RecordingID)
{
        return cRecordMediaChannel->HandleReleaseRecording(this, ct, RecordingID);
}
/************************************************/
//
// Telephony Notifications
```

```
//
/*************************************************/
CMSMessage::SerialNumberT
SessionNode::NotifyRequestCancelled(RequestToken& request)
{
  // TBD. This method will not be needed because
  // Do cancel method will return the status directly.
  Synchronized sync(this); // for future use
  RequestCancelledNotification ntfy(request);
//  return WriteWrappedMessage(ntfy);
  return 0;
}
CMSMessage::SerialNumberT
SessionNode::NotifySITDetected(CallToken& ct)
{
  Synchronized sync(this); // for future use
  SITDetectedNotification ntfy(ct);
  //return WriteWrappedMessage(ntfy);
  return 0;
}
CMSMessage::SerialNumberT
SessionNode::NotifyCNGDetected(CallToken& ct)
{
  Synchronized sync(this); // for future use
  CNGDetectedNotification ntfy(ct);
  //return WriteWrappedMessage(ntfy);
  return 0;
}
CMSMessage::SerialNumberT
SessionNode::NotifyCallAccepted(CallToken& ct)
{
  Synchronized sync(this); // for future use
  CallAcceptedNotification ntfy(ct);
  //return WriteWrappedMessage(ntfy);
  return 0;
}
CMSMessage::SerialNumberT
SessionNode::NotifyCallRejected(CallToken& ct)
{
  Synchronized sync(this); // for future use
  CallRejectedNotification ntfy(ct);
  //return WriteWrappedMessage(ntfy);
  return 0;
}
/*************************************************/
//
// Media Notifications
//
/*************************************************/
CMSMessage::SerialNumberT
SessionNode::NotifyStoppedPlaying(CallToken& callToken)
{
  Synchronized sync(this);
  StoppedPlayingNotification ntfy(callToken);
  // return WriteWrappedMessage(ntfy);
  return 0;
```

```
}
CMSMessage::SerialNumberT
SessionNode::NotifyStoppedRecording(CallToken& callToken)
{
  Synchronized sync(this); // for future use
  StoppedRecordingNotification ntfy(callToken);
  //return WriteWrappedMessage(ntfy);
  return 0;
}
/************************************************/
//
// Speech Function
//
/************************************************/
// virtual
#if 0
bool
SessionNode::HandleStartRecognition(StartRecognitionMessage *pMsg, void *userdata)
{
  Synchronized sync(this);
  if (!IsGoingAway())
  {
    SpeechChannel* sp = cSpeechChannel;
    MediaChannel* play = cPlayMediaChannel;
    MediaChannel* record = cRecordMediaChannel;

    if (sp == NULL || play == NULL || record == NULL)
      NotifySSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
                SSPF_INVALIDCALLTOKEN);
    else
      return sp->HandleStartRecognition(this, pMsg,
                          play, record, userdata);
  }
  else
    NotifySSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), SSPF_INVALIDCALLTOKEN);
  return true;
}
// virtual
bool
SessionNode::HandleStopRecognition(StopRecognitionMessage *pMsg, void *userdata)
{
  Synchronized sync(this);
  if (!IsGoingAway())
  {
    SpeechChannel* sp = cSpeechChannel;
    if (sp == NULL)
      NotifySSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
                SSPF_INVALIDCALLTOKEN);
    else
      return sp->HandleStopRecognition(this, pMsg, userdata);
  }
  else
    NotifySSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), SSPF_INVALIDCALLTOKEN);
  return false;
}
// virtual
```

```
bool
SessionNode::HandleRecognizeString(RecognizeStringMessage *pMsg, void *userdata)
{
  Synchronized sync(this);
  if (!IsGoingAway())
  {
    SpeechChannel* sp = cSpeechChannel;
    if (sp == NULL)
      NotifySSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
                SSPF_INVALIDCALLTOKEN);
    else
      return sp->HandleRecognizeString(this,
                          pMsg, userdata);
  }
  else
    NotifySSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), SSPF_INVALIDCALLTOKEN);
  return true;
}
#endif
// virtual
#if 0
bool
SessionNode::HandleCreateDynamicGrammar(CreateDynamicGrammarMessage *pMsg, void *userdata)
{
  Synchronized sync(this);
  if (!IsGoingAway())
  {
    SpeechChannel* sp = cSpeechChannel;
    if (sp == NULL)
      NotifySSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
                SSPF_INVALIDCALLTOKEN);
    else
      return sp->HandleCreateDynamicGrammar(this,
                          pMsg, userdata);
  }
  else
    NotifySSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), SSPF_INVALIDCALLTOKEN);
  return true;
}
// virtual
bool
SessionNode::HandleDeleteDynamicGrammar(DeleteDynamicGrammarMessage *pMsg, void *userdata)
{
  Synchronized sync(this);
  if (!IsGoingAway())
  {
    SpeechChannel* sp = cSpeechChannel;
    if (sp == NULL)
      NotifySSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
                SSPF_INVALIDCALLTOKEN);
    else
      return sp->HandleDeleteDynamicGrammar(this,
                          pMsg, userdata);
  }
  else
    NotifySSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), SSPF_INVALIDCALLTOKEN);
```

```
    return true;
}
// virtual
bool
SessionNode::HandleAddPhraseToDynamicGrammar(AddPhraseToDynamicGrammarMessage *pMsg, void
*userdata)
{
    Synchronized sync(this);
    if (!IsGoingAway())
    {
        SpeechChannel* sp = cSpeechChannel;
        if (sp == NULL)
            NotifySSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
                    SSPF_INVALIDCALLTOKEN);
        else
            return sp->HandleAddPhraseToDynamicGrammar(this, pMsg, userdata);
    }
    else
        NotifySSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), SSPF_INVALIDCALLTOKEN);
    return true;
}
// virtual
bool
SessionNode::HandleRemovePhraseFromDynamicGrammar(RemovePhraseFromDynamicGrammarMessage *pMsg,
void *userdata)
{
    Synchronized sync(this);
    if (!IsGoingAway())
    {
        SpeechChannel* sp = cSpeechChannel;
        if (sp == NULL)
            NotifySSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
                    SSPF_INVALIDCALLTOKEN);
        else
            return sp->HandleRemovePhraseFromDynamicGrammar(this, pMsg, userdata);
    }
    else
        NotifySSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), SSPF_INVALIDCALLTOKEN);
    return true;
}
// virtual
bool
SessionNode::HandleQueryDynamicGrammarContents(QueryDynamicGrammarContentsMessage *pMsg, void
*userdata)
{
    Synchronized sync(this);
    if (!IsGoingAway())
    {
        SpeechChannel* sp = cSpeechChannel;
        if (sp == NULL)
            NotifySSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
                    SSPF_INVALIDCALLTOKEN);
        else
            return sp->HandleQueryDynamicGrammarContents(this, pMsg, userdata);
    }
    else
```

```
    NotifySSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), SSPF_INVALIDCALLTOKEN);
  return true;
}
// virtual
bool
SessionNode::HandleInsertDynamicGrammar(InsertDynamicGrammarMessage *pMsg, void *userdata)
{
  Synchronized sync(this);
  if (!IsGoingAway())
  {
    SpeechChannel* sp = cSpeechChannel;
    if (sp == NULL)
      NotifySSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(),
            SSPF_INVALIDCALLTOKEN);
    else
      return sp->HandleInsertDynamicGrammar(this, pMsg, userdata);
  }
  else
    NotifySSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), SSPF_INVALIDCALLTOKEN);
  return true;
}
#endif
/*************************************************/
//
// Notification from SSP
//
/*************************************************/
#if 0
CMSMessage::SerialNumberT
SessionNode::NotifyBeginSpeech(CallToken& ct)
{
  Synchronized sync(this); // for future use
  BeginSpeechNotification ntfy(ct);
  return WriteWrappedMessage(ntfy);
}
CMSMessage::SerialNumberT
SessionNode::NotifyEndSpeech(CallToken& ct)
{
  Synchronized sync(this); // for future use
  EndSpeechNotification ntfy(ct);
  return WriteWrappedMessage(ntfy);
}
#endif
CMSMessage::SerialNumberT
SessionNode::NotifyRecognitionStarted(CallToken& ct)
{
  Synchronized sync(this); // for future use
  RecognitionStartedNotification ntfy(ct);
  //return WriteWrappedMessage(ntfy);
  return 0;
}
#if 0
CMSMessage::SerialNumberT
SessionNode::NotifyDynamicGrammarDeleted(CallToken& ct,
                DynamicGrammarResultNotification::Result result, const gm::string& db_key)
{
```

```
  Synchronized sync(this); // for future use
  DeleteDynamicGrammarResultNotification ntfy(ct, db_key, result);
  //return WriteWrappedMessage(ntfy);
  return 0;
}
CMSMessage::SerialNumberT
SessionNode::NotifyPhraseAddedToDynamicGrammar(CallToken& ct,
                                DynamicGrammarResultNotification::Result result, const gm::string& db_key,
        const gm::string& phrase_id)
{
  Synchronized sync(this); // for future use
  AddPhraseToDynamicGrammarResultNotification ntfy(ct, db_key, result, phrase_id);
  //return WriteWrappedMessage(ntfy);
  return 0;
}
CMSMessage::SerialNumberT
SessionNode::NotifyPhraseRemovedFromDynamicGrammar(CallToken& ct,
                                  DynamicGrammarResultNotification::Result result, const gm::string& db_key,
        const gm::string& phrase_id)
{
  Synchronized sync(this); // for future use
  RemovePhraseFromDynamicGrammarResultNotification ntfy(ct, db_key, result, phrase_id);
  //return WriteWrappedMessage(ntfy);
  return 0;
}
#endif
/**
 * Dropping Call
 *
 */
void
SessionNode::NotifyAboutToDropCall(CallToken& ct)
{
  Synchronized sync(this);
  SetGoingAway(true);
  MediaChannel* play = cPlayMediaChannel;
  MediaChannel* record = cRecordMediaChannel;
  SpeechChannel* speech = cSpeechChannel;
  if (play != NULL)
    play->NotifyCallAboutToBeDropped(this, ct);
  if (record != NULL)
    record->NotifyCallAboutToBeDropped(this, ct);
  if (speech != NULL)
    speech->NotifyCallAboutToBeDropped(this, ct);
}
/**
 * Log Item
 */
bool
SessionNode::HandleLogEnterState(const gm::string& StateName)
{
  Synchronized sync(this); // for future use
  if (IsGoingAway() == false) {
    return cSessionLogger.NotifyEnteringState(StateName);
  } else {
    return false;
```

```
    }
  }
  bool
  SessionNode::HandleLogItem(const gm::string& ItemName,
                             const gm::string& ItemValue) {
    Synchronized sync(this); // for future use
    if (IsGoingAway() == false) {
      return cSessionLogger.LogItem(ItemName, ItemValue);
    } else {
      return false;
    }
  }
  bool
  SessionNode::HandleLogLeaveState() {
    Synchronized sync(this); // for future use
    if (IsGoingAway() == false) {
      return cSessionLogger.NotifyLeaveState();
    } else {
      return false;
    }
  }
```

Document Number: 2

```
/*
 * SetPropertyMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:21:57 $
 */
static const char* id="$Id: SetPropertyMsg.cpp,v 1.5 2000/12/07 03:21:57 Mitsuru Oshima Exp $";
#include "SetPropertyMsg.h"
#if 0
#include "VMachineID.h"
// virtual
VirtualMachineID&
SetPropertyMessage::GetMTSVirtualMachineID(void)
{
        static VirtualMachineID broadcast;
        broadcast.SetBroadcast(true);
        return broadcast;
}
#endif
```

```
/*
 * SilenceSegment.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:19:53 $
 */
static const char* id="$Id: SilenceSegment.cpp,v 1.5 2000/12/07 03:19:53 Mitsuru Oshima Exp $";
#include "SilenceSegment.h"
#include "MediaChannel.h"
// virtual
SilenceSegment::~SilenceSegment(void)
{
}
// virtual
bool
SilenceSegment::Play(MediaChannel *pMediaChannel,
                     CallToken& ct)
{
        //::Sleep(cDuration);
        return (bool)(pMediaChannel->DoPlaySilence(ct, cDuration) == MSPF_OK);
}
```

```
/*
 * Slot.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.6 $
 * $Date: 2000/12/07 03:21:58 $
 */
static const char* id="$Id: Slot.cpp,v 1.6 2000/12/07 03:21:58 Mitsuru Oshima Exp $";
#include "Slot.h"
#include "VPPMsg.h"
// virtual
Slot::~Slot(void)
{
        std::list<Slot *>::iterator iter = cStructureValue.begin();
        while (iter != cStructureValue.end())
        {
                delete *iter;
                iter++;
        }
}
void
Slot::AddStructSlotValue(Slot *pSlot)
{
        cType = T_Structure;
        cStructureValue.push_back(pSlot);
}
Slot *
Slot::GetStructSlot(int index)
{
        int i = 0;
        std::list<Slot *>::iterator iter = cStructureValue.begin();
        while (i < index && iter != cStructureValue.end())
        {
                iter++;
                i++;
        }
        if (iter == cStructureValue.end())
                return 0;
        return *iter;
}
void
Slot::Serialize(VPPMessage *pMsg)
{
        pMsg->InsertString(cName);
        pMsg->InsertUshort((unsigned short)cType);
        // pMsg->InsertFloat(cConfidence);
        switch (cType)
        {
                case T_String:
                        pMsg->InsertString(cStringValue);
                        break;
```

```
                case T_Int:
                        pMsg->InsertUlong((unsigned long)cIntValue);
                        break;
                case T_Structure:
                        {
                                pMsg->InsertUshort((unsigned short)cStructureValue.size());
                                std::list<Slot *>::iterator iter = cStructureValue.begin();
                                while (iter != cStructureValue.end())
                                {
                                        (*iter)->Serialize(pMsg);
                                        iter++;
                                }
                        }
                        break;
        }
}
void
Slot::Deserialize(VPPMessage *pMsg)
{
        pMsg->ExtractString(cName);
        cType = (Type)pMsg->ExtractUshort();
        //cConfidence = pMsg->ExtractFloat();
   cConfidence = 1.0;
        switch (cType)
        {
                case T_String:
                        pMsg->ExtractString(cStringValue);
                        break;
                case T_Int:
                        cIntValue = (int)pMsg->ExtractUlong();
                        break;
                case T_Structure:
                        {
                                unsigned short cnt = pMsg->ExtractUshort();
                                while (cnt--)
                                {
                                        Slot *pSlot = new Slot;
                                        pSlot->Deserialize(pMsg);
                                        AddStructSlotValue(pSlot);
                                }
                        }
                        break;
        }
}
```

```
/*
 * SpeechChannel.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.75 $
 * $Date: 2000/12/07 03:19:54 $
 */
static const char* id="$Id: SpeechChannel.cpp,v 1.75 2000/12/07 03:19:54 Mitsuru Oshima Exp $";
#define _WIN32_WINNT 0x0400
#include <windows.h>
#include <process.h>
#include "SpeechChannel.h"
#include "SpeechServiceProvider.h"
#include "SessionMgr.h"
#include "MediaChannel.h"
#include "SessionNode.h"
#include "VPPMsg/StartRecognitionMsg.h"
#include "VPPMsg/StopRecognitionMsg.h"
#include "VPPMsg/RecognizeStringMsg.h"
#include "VPPMsg/RecognitionResults.h"
#include "VPPUtils/Synchronized.h"
#include "PromptSegment.h"
#include "VPPMsg/Property.h"
#define BEGINSPEECHTIMEOUTDEFAULT 3000
#define TOOMUCHSPEECHTIMEOUTDEFAULT 5000
#define DTMFINTERDIGITTIMEOUTDEFAULT 1000
SpeechChannel::SpeechChannel(ServiceProvider *pServiceProvider, const char *logType, int id, bool
RequiresSettlement)
        : Channel(pServiceProvider, Channel::CT_Speech, logType, id, RequiresSettlement),
          cBargeInAllowed(false), cMagicWord(false), BOS_TimerID(-1), TMS_TimerID(-1), DTMF_TimerID(-1),
cLastGrammar()
{
        cState = S_Idle;
}
// virtual
SpeechChannel::~SpeechChannel(void)
{
  KillTimers();
}
void
SpeechChannel::BOS_Timeout(void) {
  diag("BOS Timer went off");
  // BOS Timeout timer went off - abort the recognition and inform the client
  // that there was a BeginSpeechTimeout
  //
  SessionNode* session = cSession;
  if (session == NULL) {
    return;
  }
  Synchronized sync(session);
  BOS_TimerID = -1;
```

```
    if (GetState() == S_RecognizingPreSpeech)
    {
      diag("Aborting recognition due to BOS Timeout");
      AbortRecognition();
      NotifyRecognitionDone(0, RecognitionDoneNotification::RS_NoSpeech);
    }
}
void
SpeechChannel::TMS_Timeout(void) {
    diag("TMS Timer went off");
    // TMS Timeout timer went off - abort the recognition and inform the client that
    // there was too much speech
    //
    SessionNode* session = cSession;
    if (session == NULL) {
      return;
    }
    Synchronized sync(session);
    TMS_TimerID = -1;
    if (GetState() == S_RecognizingSpeech)
    {
      diag("Aborting recognition due to TMS Timeout");
      AbortRecognition();
      NotifyRecognitionDone(0, RecognitionDoneNotification::RS_TooMuchSpeech);
    }
}
void
SpeechChannel::DTMF_Timeout(void) {
    diag("DTMF Timer went off");
    SessionNode* session = cSession;
    if (session == NULL) {
      return;
    }
    Synchronized sync(session);
    DTMF_TimerID = -1;
    // DTMF interdigit timer went off
    //
    if (GetState() == S_RecognizingDTMF)
      EndDTMFRecognition();
}
void
SpeechChannel::EndDTMFRecognition(void)
{
        diag("Ending DTMF recognition");
        SetState(S_RecognizingPostDTMF);
        RecognizeString(cLastGrammar, cDTMFString);
}
unsigned long
SpeechChannel::DoStartRecognition(const GrammarHandle& handle)
{
        unsigned long tGotStartRec = ::GetTickCount(); // debug
        unsigned long ret;
        diag("Got DoStartRecognition() in state %d.", cState);
        ret = StartRecognition(handle);
        unsigned long tAfterStartRec = ::GetTickCount();
        if (ret == SSPF_OK)
```

```
        {
          diag("Tell recserver to start recognition - 1delay = %.3f.", (float)((tAfterStartRec-tGotStartRec)/1000.0));
                gm::string activeGrammarNames;
                cSession->cProperties.GetProperty("Speech.ActiveGrammarNames", activeGrammarNames);
                if (*activeGrammarNames.c_str())
                        cSession->cSessionLogger.LogItem("GRAMMAR", activeGrammarNames);
                else
                        cSession->cSessionLogger.LogItem("GRAMMAR",
                                handle.key.c_str());
//              gm::string bargein = cBargeInAllowed ? "TRUE" : "FALSE";
//              cSession->cSessionLogger.LogItem("BARGE_IN", bargein);
                SetState(S_RecognizingPreSpeech);
                cLastGrammar = handle;
                if (cSession->cPlayMediaChannel)
                        cSession->cPlayMediaChannel->RequestNotification(this);
                if (cSession->cRecordMediaChannel)
                        cSession->cRecordMediaChannel->RequestNotification(this);
                if (cSession->cRecordMediaChannel && cSession->cRecordMediaChannel-
>SetSampleListener(this))
                {
                        diag("Telling media channel <%d> to start transmission.", cSession-
>cRecordMediaChannel->GetID());
                        if (!cSession->cRecordMediaChannel->StartTransmission())
                        {
                                ret = SSPF_CANTRECORD;
                                AbortRecognition();
                                warning("Can't record because StartTransmission() failed.");
                        }
                        else
                        {
                                diag("Told media channel <%d> to start transmission.", cSession-
>cRecordMediaChannel->GetID());
                                cSession->cRecordMediaChannel->GetRecordingFilename(cRecordingFilename);
// MT 08-18-99 - just for testing/debugging purpose
                                if (cSession->cPlayMediaChannel &&
                                        cSession->cPlayMediaChannel->GetState() == MediaChannel::S_Idle)
                                {
                                        diag("Starting BOS timer.");
                                        StartBOSTimer();
                                        diag("Started BOS timer.");
                                }
                                if (!cRound)
                                {
                                        info("Notifying recognition started.");
                                        cSession->NotifyRecognitionStarted(cCurrentCallToken);
                                        diag("Notified recognition started.");
                                        unsigned long tNotifyRecStarted = ::GetTickCount();
                                        debug("Notified recognition started - 2delay = %.3f.",
(float)((tNotifyRecStarted-tAfterStartRec)/1000.0));
                                }
                                cRound++;
                                ret = SSPF_OK;
                        }
                }
                else
                {
```

```
                        StopRecognition();
                        warning("Can't record because RecordMediaChannel does not exist or SetSampleListener()
failed.");
                        ret = SSPF_CANTRECORD;
                }
        }
        return ret;
}
void
SpeechChannel::NotifyStartOfSpeech(void)
{
        diag("Got NotifyStartOfSpeech()");
    SessionNode* session = cSession;
    if (session == NULL) {
      return;
    }
    Synchronized sync(session);
        // no need to check for session->IsGoingAway() <-- why? M.O
        if (session && GetState() == S_RecognizingPreSpeech)
        {
                SetState(S_RecognizingSpeech);
                long value = 1;
                session->cProperties.GetProperty("Speech.StopPlayOnBargeIn", value);
                gm::string bargein = "FALSE";
                if (value && session->cPlayMediaChannel)
                {
#ifdef ENABLE_PERF_MEASURE
                        // timing
                        session->cPlayMediaChannel->SetDoStopTime(cBOSTime);
#endif
                        if (session->cPlayMediaChannel->GetState() == MediaChannel::S_Playing ||
                                session->cPlayMediaChannel->GetState() == MediaChannel::S_PlayingTTS )
                        {
                                bargein = "TRUE";
                        }
                        session->cPlayMediaChannel->DoStop();
                }
                session->cSessionLogger.LogItem("BARGE_IN", bargein);
#ifdef NOTIFY_ALL
                cSession->NotifyBeginSpeech(cCurrentCallToken);
#endif
                // Stop the BOS Timeout timer
                //
         StopTimer(BOS_TimerID);
                // SetTimer(cObjects[0], 0);
                // Start the TMS Timeout timer
                //
                StartTMSTimer();
        }
}
void
SpeechChannel::NotifyEndOfSpeech(void)
{
        diag("Got NotifyEndOfSpeech()");
    SessionNode* session = cSession;
    if (session == NULL) {
```

```
        return;
    }
    Synchronized sync(session);
        // no need to check for session->IsGoingAway()
        if (GetState() == S_RecognizingSpeech)
        {
                // Caller stopped speaking - end the recognition and get the result
                //
                if (session && session->cRecordMediaChannel)
                        session->cRecordMediaChannel->StopTransmission();
                if (session)
                {
#ifdef NOTIFY_ALL
                        diag("Calling SSP->NotifyEndSpeech");
                        cSession->NotifyEndSpeech(cCurrentCallToken);
                        diag("Called SSP->NotifyEndSpeech");
#else
                        diag("Got EndSpeech event");
#endif
                }
                SetState(S_PostEndOfSpeech);
                StopRecognition();
        }
}
void
SpeechChannel::DumpSlot(Slot *pSlot, gm::string& str)
{
        char buf[20];
        str += "<";
        str += pSlot->GetName();
        str += " ";
        switch (pSlot->GetType())
        {
                case Slot::T_String:
                        str += pSlot->GetString();
                        break;
                case Slot::T_Int:
                        ::itoa(pSlot->GetInt(), buf, 10);
                        str += buf;
                        break;
                case Slot::T_Structure:
                        {
                                int i = 0;
                                Slot *pSubSlot;
                                while ((pSubSlot = pSlot->GetStructSlot(i++)))
                                        DumpSlot(pSubSlot, str);
                        }
                        break;
        }
        str += "> ";
}
void
SpeechChannel::NotifyRecognitionDone(RecognitionResults *pResults,
        RecognitionDoneNotification::RecognitionStatus status)
{
   diag("Got NotifyRecognitionDone");
```

```
SessionNode* session = cSession;
if (session == NULL) {
  return;
}
Synchronized sync(session);
// no need to check for session->IsGoingAway()
if (status == RecognitionDoneNotification::RS_SuccessfulSpeech)
  LogUtteranceFilename();
char buf[100];
unsigned short results = pResults ? pResults->GetNumResults() : 0;
StopTimer(TMS_TimerID);
if (status == RecognitionDoneNotification::RS_SuccessfulSpeech ||
  status == RecognitionDoneNotification::RS_SuccessfulDTMF)
{
  if (session)
    session->cSessionLogger.LogItem("NUM_RESULTS", results);

  for (unsigned short result = 0; result < results; result++)
  {
    RecognitionResult *pResult = pResults->GetResult(result);

    ::sprintf(buf, "RESULT[%d]", result);

    if (status == RecognitionDoneNotification::RS_SuccessfulDTMF)
      pResult->SetResult(cDTMFString);

    if (session)
      session->cSessionLogger.LogItem(buf, pResult->GetResult());

    ::sprintf(buf, "CONFIDENCE[%d]", result);
    if (session)
      session->cSessionLogger.LogItem(buf, (unsigned short)(pResult->GetConfidence() * 100));

    unsigned short interpretations = pResult->GetNumNLInterpretations();

    ::sprintf(buf, "NUM_NL_INTERPRETATIONS[%d]", result);
    if (session)
      session->cSessionLogger.LogItem(buf, interpretations);

    for (unsigned short interp = 0; interp < interpretations; interp++)
    {
      NLInterpretation *pNLInterp = pResult->GetNLInterpretation(interp);

      unsigned short slots = pNLInterp->GetNumSlots();

      gm::string nlresultstring("{");

      for (unsigned short slot = 0; slot < slots; slot++)
        DumpSlot(pNLInterp->GetSlot(slot), nlresultstring);

      nlresultstring += "}";

      ::sprintf(buf, "NL_INTERPRETATION[%d][%d]", result, interp);
      if (session)
        session->cSessionLogger.LogItem(buf, nlresultstring);
    }
```

```
    }
  }
  else
  {
    gm::string resultstring;

    if (status == RecognitionDoneNotification::RS_NoSpeech)
    {
      resultstring = "<no_speech_timeout>";
      diag("No speech timeout.");
    }
    else if (status == RecognitionDoneNotification::RS_TooMuchSpeech)
    {
      resultstring = "<too_much_speech_timeout>";
    }

    if (resultstring)
    {
      if (session)
      {
        session->cSessionLogger.LogItem("NUM_RESULTS", (unsigned short)1);
        session->cSessionLogger.LogItem("RESULT[0]", resultstring);
        session->cSessionLogger.LogItem("CONFIDENCE[0]", (unsigned short)100);
        session->cSessionLogger.LogItem("NUM_NL_INTERPRETATIONS[0]", (unsigned short)0);
      }
    }
  }

  if (GetState() != S_RecognizingDTMF)
  {
    SetState(S_Idle);
    KillTimers();

    // If we're doing magic word recognition, and we have no results, we just
    // want to start the recognition all over again
    //
    if (cMagicWord && (!pResults || !pResults->GetNumResults() ||
                !pResults->GetResult(0)->GetNumNLInterpretations()))
    {
      DoStartRecognition(cLastGrammar);
      return;
    }
    // RecognitionDoneNotification ntfy(cCurrentCallToken, status, pResults);
    diag("Calling SSP->NotifyRecognitionDone");
    cSession->NotifyRecognitionDone(cCurrentCallToken,
                        pResults,
                        status);
    diag("Called SSP->NotifyRecognitionDone, calling RecordMediaChannel::StopTransmission() and
RequestNotification()");

    if (session && session->cRecordMediaChannel)
    {
      session->cRecordMediaChannel->StopTransmission();
      session->cRecordMediaChannel->RequestNotification(0);
    }
```

```
    debug("Called RecordMediaChannel::StopTransmission(), calling PlayMediaChannel::DoStop() and
RequestNotification()");
    if (session && session->cPlayMediaChannel)
    {
      session->cPlayMediaChannel->DoStop();
      session->cPlayMediaChannel->RequestNotification(0);
    }
    debug("Called PlayMediaChannel::DoStop() and RequestNotification()");
  }
  else
  {
    if (session && session->cPlayMediaChannel)
    {
      session->cPlayMediaChannel->DoStop();
      session->cPlayMediaChannel->RequestNotification(0);
    }

    if (session && session->cRecordMediaChannel)
    {
      session->cRecordMediaChannel->StopTransmission();
      session->cRecordMediaChannel->RequestNotification(0);
    }
  }
}
void
SpeechChannel::NotifyPlayStarted(void)
{
        if (GetState() == S_RecognizingPreSpeech)
        {
                // Stop the BOS timer
                //
                diag("Stopping BOS timer due to PlayStarted");
    StopTimer(BOS_TimerID);
                // SetTimer(cObjects[0], 0);
        }
}
void
SpeechChannel::NotifyPlayDone(void)
{
  if (GetState() == S_RecognizingPreSpeech)
  {
    // Start the BOS timer
    //
    diag("Starting BOS timer due to PlayDone");
    StartBOSTimer();
  }
}
void
SpeechChannel::NotifyPlayStopped(void)
{
  if (GetState() == S_RecognizingPreSpeech)
  {
    // Start the BOS timer
    //
    diag("Starting BOS timer due to PlayStopped");
    StartBOSTimer();
```

```
    }
}
void
SpeechChannel::AppendDTMFChar(char which)
{
        cDTMFString += "dtmf-";
        if (which == '#')
                cDTMFString += "pound";
        else if (which == '*')
                cDTMFString += "star";
        else
                cDTMFString += which;
        cDTMFString += " ";
        if (++cDTMFDigits >= cDTMFMaxDigits && cDTMFMaxDigits)
                EndDTMFRecognition();
}
void
SpeechChannel::NotifyDTMF(char which)
{
        info("Got DTMF-%c", which);
    SessionNode* session = cSession;
    if (session == NULL) {
      return;
    }
    Synchronized sync(session);
        // no need to check for session->IsGoingAway()
        if (GetState() == S_RecognizingPreSpeech || GetState() == S_RecognizingSpeech)
        {
                diag("Transitioning to S_RecognizingDTMF");
                // Go into DTMF recognition mode
                //
                SetState(S_RecognizingDTMF);
                if (session->cPlayMediaChannel)
                        session->cPlayMediaChannel->DoStop();
                // Abort the speech recognition
                //
                AbortRecognition();
                long value = 0;
                session->cProperties.GetProperty("Speech.DTMFMaxDigits", value);
                cDTMFMaxDigits = value;
                cDTMFDigits = 0;
                cDTMFString = "";
                AppendDTMFChar(which);
                // Start the dtmf timer
                //
                StartDTMFTimer();
        }
        else if (GetState() == S_RecognizingDTMF)
        {
                // Reset the DTMF timer
                //
                StartDTMFTimer();
                // add to the string
                //
                AppendDTMFChar(which);
                diag("DTMF String is now %s", cDTMFString.c_str());
```

```
        }
}
SpeechChannel::ChannelState
SpeechChannel::GetState(void)
{
        Synchronize sync(cCriticalSection);
        return cState;
}
void
SpeechChannel::SetState(SpeechChannel::ChannelState state)
{
  diag("Changing state from %d to %d", cState, state);
        Synchronize sync(cCriticalSection);
        cState = state;
  diag("Changed state from %d to %d", cState, state);
}
void
SpeechChannel::KillTimers(void)
{
  StopTimer(BOS_TimerID);
  StopTimer(TMS_TimerID);
  StopTimer(DTMF_TimerID);
}
void
SpeechChannel::StopTimer(Timer::TimerID &id) {
  if (id>=0) {
    GetSSP()->GetTimer().remove(id);
    id = -1;
  }
}
// virtual
void
SpeechChannel::NotifySample(const unsigned char *pSample, unsigned int numBytes)
{
  if (GetState() == S_Idle || GetState() == S_PostEndOfSpeech) {
    debug("Discarding sample : state = %d", GetState());
                return;
  }
        info("Got NotifySample");
//      HANDLE hackSomeEvent = ::CreateEvent(NULL, false, true, NULL);
//      SessionLock session(this, (unsigned long)hackSomeEvent);
//      ::CloseHandle(hackSomeEvent);

    SessionNode* session = cSession;
    if (session == NULL) {
      return;
    }
    Synchronized sync(session);
        if (session->IsGoingAway())
                return;
//      if (!cSession)  // tmt locking changes
//              return;
        // Only push the samples if barge-in is allowed, or we are finished playing
        //
        if (cBargeInAllowed ||
```

```
                (session->cPlayMediaChannel && session->cPlayMediaChannel->GetState() ==
MediaChannel::S_Idle)) {
                PushSamples(pSample, numBytes);
                return;
        }
        else {
                diag("Bitbucketing samples because barge-in is not allowed");
                return;
        }
}
// virtual
void
SpeechChannel::NotifyEndOfData(int reason)
{
        if (reason == -1) {
                // notify failure
       cSession->NotifySSPFailure(cCurrentCallToken, cMsgSerialNumber, SSPF_CANTRECOGNIZE);
        }
}
// virtual
void
SpeechChannel::NotifyPropertySet(Property& prop)
{
        diag("Had property %s set to %s",
                prop.GetName().c_str(), prop.GetValue().c_str());
}
void
SpeechChannel::StartBOSTimer(void)
{
  SessionNode* session = cSession;
  if (session == NULL) {
    return;
  }
  Synchronized sync(session);

  if (session->IsGoingAway())
    return;

  if (!cMagicWord)
  {
    long timeout = BEGINSPEECHTIMEOUTDEFAULT;
    session->cProperties.GetProperty("Speech.BeginSpeechTimeout", timeout);
    diag("Setting BeginSpeechTimeout to %ld", timeout);

    TimerEvent* ev =
      new TimerEventT<SpeechChannel>(timeout, this, SpeechChannel::BOS_Timeout);
    BOS_TimerID = GetSSP()->GetTimer().add(ev);
  }
}
void
SpeechChannel::StartTMSTimer(void)
{
  SessionNode* session = cSession;
  if (session == NULL) {
    return;
  }
```

```
    Synchronized sync(session);
    if (session->IsGoingAway())
        return;
    long timeout = TOOMUCHSPEECHTIMEOUTDEFAULT;
    session->cProperties.GetProperty("Speech.TooMuchSpeechTimeout", timeout);
    diag("Setting TooMuchSpeechTimeout to %ld", timeout);
    TimerEvent* ev =
        new TimerEventT<SpeechChannel>(timeout, this, SpeechChannel::TMS_Timeout);
    TMS_TimerID = GetSSP()->GetTimer().add(ev);
}
void
SpeechChannel::StartDTMFTimer(void)
{
    SessionNode* session = cSession;
    if (session == NULL) {
        return;
    }
    Synchronized sync(session);
    if (session->IsGoingAway())
        return;
    long timeout = DTMFINTERDIGITTIMEOUTDEFAULT;
    session->cProperties.GetProperty("Speech.DTMFInterdigitTimeout", timeout);
    diag("Setting DTMFInterdigitTimer to %ld", timeout);
    if (DTMF_TimerID != -1) {
        GetSSP()->GetTimer().remove(DTMF_TimerID);
    }
    TimerEvent* ev =
        new TimerEventT<SpeechChannel>(timeout, this, SpeechChannel::DTMF_Timeout);
    DTMF_TimerID = GetSSP()->GetTimer().add(ev);
}
bool
SpeechChannel::NotifyCallAboutToBeDropped(SessionNode *pSession, CallToken& callToken)
{
        if (GetState() != S_Idle)
        {
                SetState(S_Idle);
                info("Aborting recognition due to hangup");
                AbortRecognition();
                diag("Aborted recognition due to hangup");
                NotifyRecognitionDone(0, RecognitionDoneNotification::RS_CallDropped);
        }
        return true;
}
// virtual
bool
SpeechChannel::NotifyStartLogging(void)
{
        return StartLogging();
}
// virtual
bool
SpeechChannel::NotifyStopLogging(void)
{
        return StopLogging();
}
// new speech API
```

```
unsigned long
SpeechChannel::HandleStartRecognition(SessionNode *pSession,
                        const GrammarHandle& handle,
                        MediaChannel *pPlayChannel,
                        MediaChannel *pRecordChannel)
{
  if (!VerifySession(pSession))
    return false;
  unsigned long ret;
  if (!cSession->cRecordMediaChannel) {
    error("no record media channel");
    ret = SSPF_INSUFFICIENTRESOURCES;
  } else if (cSession->cRecordMediaChannel->GetState() == MediaChannel::S_Idle)
  {
    long value = 1;
    cSession->cProperties.GetProperty("Speech.AllowBargeIn", value);
    cBargeInAllowed = (bool)(value != 0);
    cMagicWord = false;
    cRound = 0;
    ret = DoStartRecognition(handle);
  }
  else {
    error("HandleStartRecognition: RecordMedia channel is not idle %d", cSession->cRecordMediaChannel-
>GetState());
    ret = SSPF_RECORDCHANNELNOTIDLE;
  }

  return ret;
}
bool
SpeechChannel::HandleStopRecognition(SessionNode *pSession, StopRecognitionMessage *pMsg, void *userdata)
{
        if (!VerifySession(pSession))
                return false;
        unsigned long ret = StopRecognition();
        if (ret == SSPF_OK)
        {
                // The SpeechChannel expects to get notifications from the subclass regarding the
                // end of recognition, just as it would have gotten had the recognizer detected
                // end of speech.  The only thing we do here is kill the timers so that we don't
                // get a bogus timeout while we're waiting for those results
                //
                KillTimers();
                return true;
        }
        pSession->NotifySSPFailure(pMsg->GetCallToken(), pMsg->GetSerialNo(), ret);
  return false;
}
unsigned long
SpeechChannel::HandleRecognizeString(SessionNode *pSession,
                        const GrammarHandle& handle,
                        const gm::string& str)
{
  if (!VerifySession(pSession))
    return false;
```

```
    unsigned long ret = RecognizeString(handle, str);
    if (ret == SSPF_OK)
    {
        SetState(S_RecognizingString);
        cLastGrammar = handle;
    }

    return ret;
}
void
SpeechChannel::NotifySettled(void)
{
        info("SpeechChannel::NotifySettled, cState = %d", cState);
        Synchronized sync(SessionMgr::GetSessionMgr());
    Channel::NotifySettled();
        sync.notifyAll();
}
```

Docket No. 50277-2096

```
/*
 * SpeechServiceProvider.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.28 $
 * $Date: 2000/12/07 03:19:54 $
 */
static const char* id="$Id: SpeechServiceProvider.cpp,v 1.28 2000/12/07 03:19:54 Mitsuru Oshima Exp $";
#include "SpeechServiceProvider.h"
#include "SpeechChannel.h"
#include "TelephonyServiceProvider.h"
#include "MTSServer.h"
#if 0
#include "MTSServerNode.h"
#endif
#include "VPPMsg/StartRecognitionMsg.h"
#include "VPPMsg/StopRecognitionMsg.h"
#include "VPPMsg/FailureNtfy.h"
#include "VPPMsg/RecognitionDoneNtfy.h"
#include "VPPMsg/RecognizeStringMsg.h"
#ifdef NOTIFY_ALL
#include "VPPMsg/BeginSpeechNtfy.h"
#include "VPPMsg/EndSpeechNtfy.h"
#endif
#include "VPPMsg/Property.h"
#include "VPPMsg/RecognitionStartedNtfy.h"
#include "VPPMsg/CreateDynamicGrammarMsg.h"
#include "VPPMsg/DeleteDynamicGrammarMsg.h"
#include "VPPMsg/AddPhraseToDynamicGrammarMsg.h"
#include "VPPMsg/RemovePhraseFromDynamicGrammarMsg.h"
#include "VPPMsg/QueryDynamicGrammarContentsMsg.h"
#include "VPPMsg/InsertDynamicGrammarMsg.h"
#include "VPPMsg/CreateDynamicGrammarResultNtfy.h"
#include "VPPMsg/DeleteDynamicGrammarResultNtfy.h"
#include "VPPMsg/AddPhraseToDynamicGrammarResultNtfy.h"
#include "VPPMsg/RemovePhraseFromDynamicGrammarResultNtfy.h"
#include "VPPMsg/QueryDynamicGrammarContentsResultNtfy.h"
#include "VPPMsg/InsertDynamicGrammarResultNtfy.h"
SpeechServiceProvider::SpeechServiceProvider(MTSServer *pServer, const char *logType)
        : ServiceProvider(pServer, M_NTFY_SSPFAILURE, logType), timer()
{
        SetDefaultProperty("Speech.BeginSpeechTimeout", 3000);
        SetDefaultProperty("Speech.TooMuchSpeechTimeout", 10000);
        SetDefaultProperty("Speech.DTMFInterdigitTimeout", 1000);
        SetDefaultProperty("Speech.DTMFMaxDigits", 0);
        SetDefaultProperty("Speech.AllowBargeIn", 1);
        SetDefaultProperty("Speech.StopPlayOnBargeIn", 1);
        SetDefaultProperty("Speech.EndpointerBeginSpeechTimeout", 300);
        SetDefaultProperty("Speech.EndpointerEndSpeechTimeout", 750);
        timer.start();
}
```

```
// virtual
SpeechServiceProvider::~SpeechServiceProvider(void)
{
   timer.stop();
   Shutdown();
}
SpeechChannel *
SpeechServiceProvider::FindAvailableChannel(void)
{
        std::list<Channel *>& schannels = GetChannels();
        std::list<Channel *>::iterator schiter = schannels.begin();
        while (schiter != schannels.end())
        {
                if ((*schiter)->GetState() == Channel::S_Idle)
                {
                        return (SpeechChannel *)(*schiter);
                }
                schiter++;
        }
        return 0;
}
```

```
/*
 * StartRecognitionMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:21:58 $
 */
static const char* id="$Id: StartRecognitionMsg.cpp,v 1.5 2000/12/07 03:21:58 Mitsuru Oshima Exp $";
#include "StartRecognitionMsg.h"
// virtual
void
StartRecognitionMessage::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        InsertString(cGrammar);
        InsertUshort((unsigned short)cMagicWord);
}
// virtual
void StartRecognitionMessage::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        ExtractString(cGrammar);
        cMagicWord = (bool)(ExtractUshort() != 0);
}
```

```
/*
 * t2sspmsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:22:00 $
 */
static const char* id="$Id: t2sspmsg.cpp,v 1.2 2000/12/07 03:22:00 Mitsuru Oshima Exp $";
#include "T2SSPMsg.h"
// virtual
void
T2SSPMessage::Serialize(void)
{
        printf("Serialize T2SSPMessage\n");
        VPPMessage::Serialize();
        InsertString(cText);
        InsertUchar(initialize);
}
// virtual
void T2SSPMessage::Deserialize(void)
{
        printf("DeSerialize T2SSPMessage\n");
        VPPMessage::Deserialize();
        ExtractString(cText);
        initialize = ExtractUchar();
}
T2SSPstartMessage::T2SSPstartMessage(void)
{
}
void
T2SSPstartMessage::Serialize(void)
{
        VPPMessage::Serialize();
}
// virtual
void T2SSPstartMessage::Deserialize(void)
{
        VPPMessage::Deserialize();
}
void
T2SSPstopMessage::Serialize(void)
{
        VPPMessage::Serialize();
}
// virtual
void T2SSPstopMessage::Deserialize(void)
{
        VPPMessage::Deserialize();
}
T2SSPstopMessage::T2SSPstopMessage(void)
{
```

```
}
void
T2SSPMessageReply::Serialize(void)
{
        VPPMessage::Serialize();
        InsertUlong(csynSize);
        InsertUchar(cdataDone);
        InsertRaw((char *)csynth_buffer, csynSize);
}
// virtual
void
T2SSPMessageReply::Deserialize(void)
{
        VPPMessage::Deserialize();
        csynSize = ExtractUlong();
        cdataDone = ExtractUchar();
        if (cdataDone)
        {
                csynth_buffer = new char[csynSize];
                ExtractRaw((char *)csynth_buffer, csynSize);
        }
        else
        {
                csynth_buffer =0;
        }

}
T2SSPMessageReply::~T2SSPMessageReply(void)
{
//      delete [] csynth_buffer;
//      csynth_buffer = NULL;
}
T2SSPMessage::T2SSPMessage( const gm::string& text,unsigned char init)
{
        unsigned int      text_size = strlen(text);
        cText = text;
        initialize = init;
}
```

```
/*
 * TelephonyChannel.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.57 $
 * $Date: 2000/12/07 03:19:55 $
 */
static const char* id="$Id: TelephonyChannel.cpp,v 1.57 2000/12/07 03:19:55 Mitsuru Oshima Exp $";
#define _WIN32_WINNT 0x0400
#include <windows.h>
#include <process.h>
#include "TelephonyChannel.h"
#include "TelephonyServiceProvider.h"
#include "MTSServer.h"
#include "VPPUtils/VPPDebug.h"
#include "VPPMsg/PlaceCallMsg.h"
#include "VPPMsg/DropCallMsg.h"
#include "VPPMsg/ConnectCallsMsg.h"
#include "VPPMsg/PlaceCallOnHoldMsg.h"
#include "VPPMsg/AcceptCallMsg.h"
#include "VPPMsg/RejectCallMsg.h"
#if 0
#include "VPPMsg/AssignedClientIDNtfy.h"
#endif
#include "VPPMsg/CancelRequestMsg.h"
#include "VPPMsg/DisconnectCallsMsg.h"
#include "CMS/CMSServerNode.h"
#include "VPPMsg/Property.h"
#include "SessionNode.h"
#include "Regex/RegularExpression.h"
#include <MTSLib/MTSMonitorImpl.h>
TelephonyChannel::TelephonyChannel(ServiceProvider *pServiceProvider, const char *logType, int id, bool
RequiresSettlement)
        : SCChannel(pServiceProvider, Channel::CT_Telephony, logType, id, RequiresSettlement), timerID(-1)
{
        cState   = S_Disconnected;
        cStatus  = S_OutOfService;
        cCaps    = 0;
        cCurrentRequestToken.Invalidate();
}
// virtual
TelephonyChannel::~TelephonyChannel(void)
{
  StopTimer();
}
void
TelephonyChannel::StopTimer() {
  if (timerID>=0) {
    GetTSP()->GetTimer().remove(timerID);
    timerID = -1;
  }
```

```
}
void
TelephonyChannel::InCall_Timeout() {
   SessionNode* session = cSession;
   if (session != NULL) {
      Synchronized sync (session);
      if (!session->IsGoingAway())
      {
         if (GetState() == S_IncomingNotified)
         {
            // Drop incoming call
            //
            debug("IncomingCall Timeout Drop call");
            DoDropCall();
            GetTSP()->NotifyCallBooted(cSession);
         }
      }
   }
}
void
TelephonyChannel::OutCall_Timeout() {
   SessionNode* session = cSession;
   if (session != NULL) {
      Synchronized sync (session);
      if (!session->IsGoingAway())
      {
         if (GetState() == S_OutgoingDialed)
         {
            // Cancel outgoing call
            //
                                    info("Cancelling call due to outgoing call timeout");
            DoCancelCall();
            GetTSP()->NotifyCallBooted(cSession);
         }
      }
   }
}
unsigned long
TelephonyChannel::HandlePlaceCall(const gm::string& calledNumber,
                        const gm::string& callingNumber,
                        unsigned char *uui,
                        int uuiLength,
                        RequestToken& requestToken)
{
   unsigned long ret;
   // First, check if this call is to a restricted number
   //
   ret = PlaceCall(calledNumber,
               callingNumber,
               uui,
               uuiLength
               );
   if (ret == TSPF_OK)
   {
      SetState(S_OutgoingDialed);
      cCurrentRequestToken = requestToken;
```

```
    info("Placed call for telephony channel %p, call token span %d channel %d seq num %ld TSP private %ld",
                        this,
        cCurrentCallToken.GetSpan(),
        cCurrentCallToken.GetChannel(),
        cCurrentCallToken.GetSequenceNumber(),
        cCurrentCallToken.GetTSPPrivate());
  }

  return ret;
}
unsigned long
TelephonyChannel::DoCancelCall(void)
{
        unsigned long ret;

        if (cCurrentRequestToken.IsValid())
        {
                ret = CancelRequest();
                if (ret == TSPF_OK)
                {
                        cSession->NotifyCallProgress(cCurrentRequestToken,
CallProgressNotification::CPN_NoAnswer);
                        //cSession->NotifyRequestCancelled(cCurrentRequestToken);
                        ResetChannel();
                }
        }
        else
                ret = TSPF_INVALIDREQUESTTOKEN;
        return ret;
}
/* ************************ NEW API ************************** */
bool
TelephonyChannel::HandleCancelRequest(SessionNode* pSession,
                        CancelRequestMessage *pMsg, void *userdata)
{
  info("Client has requested CancelCall");
  unsigned long ret = DoCancelCall();
  if (ret != TSPF_OK)
  {
    warning("Can't cancel call - ret status=%ld.", ret);
    //pSession->NotifyTSPFailure(pMsg->GetSerialNo(), ret);
    return false;
  }
  return true;
}
bool
TelephonyChannel::HandleDropCall(SessionNode* pSession,
                        DropCallMessage *pMsg, void *userdata)
{
  info("Client has requested DropCall");
  unsigned long ret = DoDropCall();
  if (ret != TSPF_OK)
  {
    pSession->NotifyTSPFailure(pMsg->GetSerialNo(), ret);
    return false;
  }
```

```
  return true;
}
bool
TelephonyChannel::HandleAcceptCall(SessionNode* pSession,
                        AcceptCallMessage *pMsg, void *userdata)
{
  unsigned long ret;
  if (cState == S_IncomingNotified)
  {
    debug("Calling AcceptCall");
    ret = AcceptCall();
    debug("Called AcceptCall");
    if (ret == TSPF_OK)
    {
      SetState(S_Connected);
      //cClientID = pSession->GetVUIClientID();
      // print the call token info
      info("HandleAcceptCall with call token span %d channel %d seq num %ld TSP private %ld",
cCurrentCallToken.GetSpan(), cCurrentCallToken.GetChannel(), cCurrentCallToken.GetSequenceNumber(),
cCurrentCallToken.GetTSPPrivate());
      // pSession->NotifyCallAccepted(cCurrentCallToken);
      return true;
    }
    else
    {
      error("Cannot accept call");
      ret = TSPF_CANTGOOFFHOOK;
    }
  }
  else {
    error("there is no call offered");
    ret = TSPF_CALLNOTOFFERED;
  }

  pSession->NotifyTSPFailure(pMsg->GetSerialNo(), ret);
  return false;
}
bool
TelephonyChannel::HandleRejectCall(SessionNode* pSession,
                        RejectCallMessage *pMsg, void *userdata)
{
  unsigned long ret;

  if (cCurrentCallToken.IsValid())
  {
    ret = RejectCall(RC_UserBusy);

    if (ret == TSPF_OK)
    {
      ResetChannel();
      pSession->NotifyCallRejected(cCurrentCallToken);
      return true;
    }
  }
  else {
    error("HandleRejectall: CallToken is not valid");
```

```
    ret = TSPF_INVALIDCALLTOKEN;
  }

  if (ret != TSPF_OK)
    pSession->NotifyTSPFailure(pMsg->GetSerialNo(), ret);

  return false;
}
/* ************************ END OF NEW API ************************ */
unsigned long
TelephonyChannel::DoDropCall(void)
{
        unsigned long ret;

        if (cCurrentCallToken.IsValid())
        {
                NotifyAboutToDropCall();
                ret = DropCall();
                if (ret == TSPF_OK)
                {
                        NotifyCallDropped();
                        ResetChannel();
                }
        }
        else {
      error("DoDropCall: CallToken is not valid");
      ret = TSPF_INVALIDCALLTOKEN;
    }
        return ret;
}
unsigned long
TelephonyChannel::HandlePlaceCallOnHold(PlaceCallOnHoldMessage *pMsg, void *userdata)
{
        unsigned long ret = PlaceCallOnHold();
    return ret;
#if 0
        if (ret == TSPF_OK)
                //
                // DFB_TBD - do we need an S_OnHold state?  How to we get OffHold?
                //
                return true;
        NotifyFailure(pServerNode, VUIClientID, pMsg->GetSerialNo(), ret);
        return false;
#endif
}
#if 0
bool
TelephonyChannel::HandleNotifySuspended(MTSServerNode *pServerNode,
        ClientID& VUIClientID, VPPMessage *pMsg, void *userdata)
{
        // DFB_TBD
        //
        return false;
}
bool
TelephonyChannel::HandleNotifySessionShutdownComplete(VPPMessage *pMsg,
```

```
                                        void *userdata)
{
        return true;
}
#endif
void
TelephonyChannel::CleanupResources(void)
{
        if (cState == S_OutgoingDialed)
        {
                info("Cleaning up resources, cancelling call");
                DoCancelCall();
        }
        else if (cState != S_Disconnected)
        {
                info("Cleaning up resources, dropping call");
                DoDropCall();
        }
}
#if 0
bool
TelephonyChannel::HandleNotifyProcessShutdownComplete(
        ProcessShutdownCompleteNotification *pMsg, void *userdata)
{
        CleanupResources();
        return true;
}
#endif
void
TelephonyChannel::NotifyCallConnected(void)
{
        SetState(S_Connected);
        // DFB_TBD - making a copy of the list is ugly...find a better way to address the
        // fact that the list changes while we're iterating the list
        //
        std::list<SCChannel *> listcopy = cChannelsListeningToMe;
        std::list<SCChannel *>::iterator chiter = listcopy.begin();
        while (chiter != listcopy.end())
        {
                SCChannel *pChannel = *chiter;
                if ((*chiter)->GetType() == CT_Telephony)
                        // Complete the remaining half of the trombone
                        //
                        GetTSP()->GetMTSServer()->GetSessionMgr()->CreateTrombone(this,
                                (TelephonyChannel *)(*chiter), true);
                chiter++;
        }
  WaitForSessionNode();
  cSession->NotifyCallConnected(cCurrentRequestToken,
                  cCurrentCallToken);

  cCurrentRequestToken.Invalidate();
}
void
TelephonyChannel::WaitForSessionNode(void) {
  Synchronized sync(this);
```

```
    while (cSession == NULL) {
      sync.wait();
    }
}
void
TelephonyChannel::NotifySITDetected(void)
{
        cSession->NotifySITDetected(cCurrentCallToken);
}
void
TelephonyChannel::NotifyCNGDetected(void)
{
        cSession->NotifyCNGDetected(cCurrentCallToken);
}
void
TelephonyChannel::NotifyCallProgress(CallProgressNotification::CallProgressT ntfy)
{
    WaitForSessionNode();
    cSession->NotifyCallProgress(cCurrentRequestToken, ntfy);
}
void
TelephonyChannel::NotifyIncomingCall(Address& DNIS, Address& ANI, char *UUI, int length)
{
        if (!(cCaps & C_Incoming))
                warning("Bad Magic!  Incoming call presented on non-C_Incoming TelephonyChannel!");
        SetState(S_IncomingNotified);
        info("NotifyIncomingCall with call token span %d channel %d seq num %ld TSP private %ld",
cCurrentCallToken.GetSpan(), cCurrentCallToken.GetChannel(), cCurrentCallToken.GetSequenceNumber(),
cCurrentCallToken.GetTSPPrivate());
        GetTSP()->NotifyIncomingCall(cCurrentCallToken, DNIS, ANI, UUI, length);
}
void
TelephonyChannel::NotifyAboutToDropCall(void)
{
        if (cState == S_IncomingNotified)
        {
                // Remote end hung up before we could accept or reject the incoming call.
                // Notify the VUI that we are revoking our previous offer
                //
                cSession->NotifyAboutToDropCall(cCurrentCallToken);
         cSession->NotifyIncomingCallRevoked(cCurrentCallToken);
                // GetTSP()->NotifyIncomingCallRevoked(cSession,
        }
        else if (cState == S_OutgoingDialed)
        {
                // Network dropped the call before remote end could accept or reject the call.
                // Notify the VUI that there was a network error
                //
                NotifyCallProgress(CallProgressNotification::CPN_Error);
        }
        else if (cState == S_Connected)
        {
                // One end of an established call hung up.  Notify the TSP that we are about
                // to drop this call.
                //
                cSession->NotifyAboutToDropCall(cCurrentCallToken);
```

```
        }
}
void
TelephonyChannel::NotifyCallDropped(void)
{
        info("TelephonyChannel::NotifyCallDropped, cState = %d",
                cState);
        // Notify Client if its in session.
        if (cState == S_Connected)
                cSession->NotifyCallDropped(cCurrentCallToken);
        SessionMgr::GetSessionMgr()->DestroySession(this);
        ResetChannel();
        // ONLY change state to S_Disconnected here
        // NOT in ResetChannel(), because the channel can
        // only be used after the call is really dropped, i.e.
        // when we get the NotifyCallDropped() message.
        SetState(S_Disconnected);
        // NotifyChannelAvailable(); //do this in NotifySettled
}
void
TelephonyChannel::NotifySettled(void)
{
        info("TelephonyChannel::NotifySettled, channel is now available, cState = %d",
                cState);
  Channel::NotifySettled();
#if 0
        NotifyChannelAvailable();
#endif
}
void
TelephonyChannel::ResetChannel(void)
{
  cCurrentRequestToken.Invalidate();
        ResetRouting();
}
void
TelephonyChannel::ResetRouting(void)
{
        GetTSP()->GetMTSServer()->GetSessionMgr()->ResetRouting(this);
        // DFB_TBD - making a copy of the list is ugly...find a better way to address the
        // fact that the list changes while we're iterating the list
        //
        std::list<SCChannel *> listcopy = cChannelsListeningToMe;
        std::list<SCChannel *>::iterator chiter = listcopy.begin();
        while (chiter != listcopy.end())
        {
                SCChannel *pChannel = *chiter;
                if ((*chiter)->GetType() == CT_Telephony)
                        GetTSP()->GetMTSServer()->GetSessionMgr()->ResetRouting(
                                (TelephonyChannel *)(*chiter));
                chiter++;
        }
}
// virtual
void
TelephonyChannel::NotifyPropertySet(Property& prop)
```

```
{
}
void
TelephonyChannel::InitializeCurrentCallToken(void)
{
#if 0
        cCurrentCallToken.SetMTSVirtualMachineID(GetTSP()->GetMTSServer()->GetVirtualMachineID());
#endif
}
#if 0
void
TelephonyChannel::NotifyChannelAvailable(void)
{
        GetTSP()->NotifyChannelAvailable();
}
#endif
void
TelephonyChannel::SetStatusAndNotifyTSP(ChannelStatus status)
{
        SetStatus(status);
        GetTSP()->NotifyChannelStatus(status, *this);
}
void
TelephonyChannel::SetState(ChannelState state)
{
  ChannelState oldstate = cState;
  cState = state;
  if (state == S_OutgoingDialed)
  {
    // Start the outgoing drop dead timer
    //
    long value = 0;
    cServiceProvider->GetMTSServer()->GetSessionMgr()->GetDefaultProperty(
                                            "System.OutgoingCallTimeout", value);

    TimerEvent* ev =
      new TimerEventT<TelephonyChannel>(value,
                        this,
                        TelephonyChannel::OutCall_Timeout);
    timerID = GetTSP()->GetTimer().add(ev);
    // SetTimer(value);

    info("TelephonyChannel %p dialing timer set to %ld",
          this, value);
  }
  else if (state == S_IncomingNotified)
  {
    // Start the incoming drop dead timer
    //
    long value = 0;
    cServiceProvider->GetMTSServer()->GetSessionMgr()->GetDefaultProperty(
                                            "System.IncomingCallTimeout", value);

    TimerEvent* ev =
      new TimerEventT<TelephonyChannel>(value,
                        this,
```

```
                                TelephonyChannel::InCall_Timeout);
        timerID = GetTSP()->GetTimer().add(ev);
        // SetTimer(value);

        info("TelephonyChannel %p dialing timer set to %ld",
                this, value);
    }
    else if (state == S_Disconnected || state == S_Connected)
    {
        // Kill the timer
        //
        StopTimer();
        // SetTimer(0);
    }
// MTSMonitor notification
    GetTSP()->GetMTSServer()-
>PostMonitorMessage(mtsmon::WM_TSP_CHANNEL,cID,MAKELONG(GetState(),GetStatus()));
}
bool
TelephonyChannel::IsAvailable(void)
{
        return (cState == TelephonyChannel::S_Disconnected && cStatus == TelephonyChannel::S_InService);
}
```

Confidential - Property of Dialogic Corporation

```
/*
 * TelephonyServiceProvider.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.68 $
 * $Date: 2000/12/08 00:42:16 $
 */
static const char* id="$Id: TelephonyServiceProvider.cpp,v 1.68 2000/12/08 00:42:16 Dmitri Khokhlov Exp $";
#include "TelephonyServiceProvider.h"
#include "TelephonyChannel.h"
#include "MTSServerImpl.h"
#include "MTSServer.h"
#include "RangeParser.h"
#include "MediaServiceProvider.h"
#include "VPPMsg/IncomingCallNtfy.h"
#include "VPPMsg/CallProgressNtfy.h"
#include "VPPMsg/CallDroppedNtfy.h"
#include "VPPMsg/CallsConnectedNtfy.h"
#include "VPPMsg/RequestCancelledNtfy.h"
#include "VPPMsg/CallConnectedNtfy.h"
#include "VPPMsg/SITDetectedNtfy.h"
#include "VPPMsg/CNGDetectedNtfy.h"
#include "VPPMsg/CallsDisconnectedNtfy.h"
#include "VPPMsg/FailureNtfy.h"
#include "VPPMsg/IncomingCallRevokedNtfy.h"
#include "VPPMsg/AppendCompositePromptPromptMsg.h"
#include "VPPUtils/ConfigMgr.h"
#include "VPPMsg/PlaceCallMsg.h"
#include "VPPMsg/DropCallMsg.h"
#include "VPPMsg/ConnectCallsMsg.h"
#include "VPPMsg/PlaceCallOnHoldMsg.h"
#include "VPPMsg/AcceptCallMsg.h"
#include "VPPMsg/RejectCallMsg.h"
#include "VPPMsg/CancelRequestMsg.h"
#include "VPPMsg/DisconnectCallsMsg.h"
#include "VPPMsg/Property.h"
#include "VPPMsg/CallAcceptedNtfy.h"
#include "VPPMsg/CallRejectedNtfy.h"
#include "VPPMsg/CDRInfoNtfy.h"
#include "VPPUtils/VPPDebug.h"
#include "VPPUtils/Thread.h"
#include "ConnectionMgr.h"
#include "CallTokenMgr.h"
#include "VUISessionImpl.h"
#include "OutboundCallRequest.h"
#include "invui_c.h"
#define DEFAULT_INCOMING_CALL_TIMEOUT 60000
#define DEFAULT_OUTGOING_CALL_TIMEOUT 60000
TelephonyServiceProvider::TelephonyServiceProvider(MTSServer *pServer, const char *logType)
        : ServiceProvider(pServer, M_NTFY_TSPFAILURE, logType), timer()
{
```

```
        cTotalNumOutboundChans = 0L;
        cIncomingRequiresMediaChannel = false;
    timer.start();
}
// virtual
TelephonyServiceProvider::~TelephonyServiceProvider(void)
{
  Shutdown();
}
// virtual
bool
TelephonyServiceProvider::PostSetup(gm::string registryParent)
{
  cIter = cChannels.begin();
  gm::string str;
        int value = DEFAULT_INCOMING_CALL_TIMEOUT;
        if (ConfigMgr::GetString(registryParent, "IncomingCallTimeout", str))
                value = ::atoi(str.c_str());

        SetDefaultProperty("System.IncomingCallTimeout", value);
        value = DEFAULT_OUTGOING_CALL_TIMEOUT;
        if (ConfigMgr::GetString(registryParent, "OutgoingCallTimeout", str))
                value = ::atoi(str.c_str());

        SetDefaultProperty("System.OutgoingCallTimeout", value);
        if (!ConfigMgr::GetString(registryParent, "IncomingLines", str))
                return false;
        RangeParser inrange(str);
        if (!ConfigMgr::GetString(registryParent, "OutgoingLines", str))
                return false;
        RangeParser outrange(str);
        std::list<Channel *>::iterator iter = cChannels.begin();
        //unsigned short ndx = 0;
        cTotalNumOutboundChans = 0L;
        while (iter != cChannels.end())
        {
                int id = ((Channel *)(*iter))->GetID();
                if (inrange.IsSet(id) && outrange.IsSet(id)) {
                        ((TelephonyChannel *)(*iter))->SetCapabilities(TelephonyChannel::C_Incoming |
                                TelephonyChannel::C_Outgoing);
                        cTotalNumOutboundChans++;
                }
                else if (inrange.IsSet(id))
                        ((TelephonyChannel *)(*iter))->SetCapabilities(TelephonyChannel::C_Incoming);
                else if (outrange.IsSet(id)) {
                        ((TelephonyChannel *)(*iter))->SetCapabilities(TelephonyChannel::C_Outgoing);
                        cTotalNumOutboundChans++;
                }
                else
                {
                        // Channel is not incoming or outgoing...must be unused
                        //
                        ((TelephonyChannel *)(*iter))->SetCapabilities(TelephonyChannel::C_Unused);
                }
                iter++;
                //ndx++;
```

```
        }
        return true;
}
// virtual
unsigned long
TelephonyServiceProvider::HandlePlaceCall(const char* number,
                                          const char* ani,
                                                                    unsigned char *uui,
                                                                    int uuiLength,
                                          OutboundCallRequest* req,
                                          CDRInfo& cdr)
{
  Synchronize sync(cCriticalSection);
  TelephonyChannel *pChannel = FindAvailableChannelForOutgoingCall();
#ifdef TROMBONE
  TelephonyChannel *pTromboneTo = (TelephonyChannel *)FindChannelForCallToken(
                pMsg->GetTransferTarget());
#endif
  unsigned long ret;
  if (pChannel)
  {
    ret = pChannel->HandlePlaceCall(number,
                                    ani,
                                                                    uui,
                                                                    uuiLength,
                                    req->GetRequestToken());
    if (ret == TSPF_OK)
    {
      Address Target(number);
      Address CallingNumber(ani);
      // Save for CDR_LOG Target and Calling numbers
      VUISessionImpl *pSessionNode =
        cServer->GetSessionMgr()->CreateSession(pChannel,
                                    Target, CallingNumber);
      req->SetSession(pSessionNode);
                                    {
                                        Synchronized chanSync(pChannel);
                                        chanSync.notify();
                                    }
      Synchronized sync(pSessionNode);
      pSessionNode->cCDRInfo = cdr;
#if 0
      pSessionNode->SetProperties(pMsg->GetProperties());
#endif
      // moved from TelephonyChannel. M.O
      // this needs to be after sessino->SetProperties.
      // pChannel->NotifyCallProgress(CallProgressNotification::CPN_Proceeding);
#ifdef TROMBONE
      if (pTromboneTo)
        cServer->GetSessionMgr()->CreateTrombone(pTromboneTo, pChannel, false);
#endif
    }
    else {
      // moved from TelephonyChannel and resumed by M.O
      // NotifyFailure(VUIClientID, pMsg->GetCallToken(), pMsg->GetSerialNo(), ret);
      // restart TSP if we got TSPF_SYSTEMFAILURE
```

```
        if (ret == TSPF_SYSTEMFAILURE) {
          if (!cTotalNumOutboundChans)
            Restart();
        }
      }
    }
    else
    {
      ret = TSPF_NOAVAILABLECHANNEL;
    }
    return ret;
}
// virtual
unsigned long
TelephonyServiceProvider::HandleConnectCalls(ConnectCallsMessage *pMsg, void *userdata)
{
    SessionNode* session1 =
      cServer->GetSessionMgr()->FindSessionNode(pMsg->GetCallToken1());
    SessionNode* session2 =
      cServer->GetSessionMgr()->FindSessionNode(pMsg->GetCallToken2());

    if (session1 == NULL || session2 == NULL) {
/*
      NotifyFailure(VUIClientID, pMsg->GetCallToken(), pMsg->GetSerialNo(), TSPF_INVALIDCALLTOKEN);
      return false;
*/
      return TSPF_INVALIDCALLTOKEN;
    }

    Synchronized sync1(session1);
    Synchronized sync2(session2);
    if (!session1->IsGoingAway() && !session2->IsGoingAway())
    {
      if (cServer->GetSessionMgr()->CreateTrombone(session1->cTelephonyChannel,
                                    session2->cTelephonyChannel, true))
        return TSPF_OK;
      else
        return TSPF_SYSTEMFAILURE;
/*
        NotifyCallsConnected(VUIClientID, pMsg->GetCallToken1(), pMsg->GetCallToken2());
      return true;
*/
    }
    else
      return TSPF_INVALIDCALLTOKEN;
/*
      NotifyFailure(VUIClientID, pMsg->GetCallToken(), pMsg->GetSerialNo(), TSPF_INVALIDCALLTOKEN);

    return false;
*/
}
unsigned long
TelephonyServiceProvider::HandleDisconnectCalls(DisconnectCallsMessage *pMsg, void *userdata)
{
    SessionNode* session1 =
      cServer->GetSessionMgr()->FindSessionNode(pMsg->GetCallToken1());
```

```
    SessionNode* session2 =
      cServer->GetSessionMgr()->FindSessionNode(pMsg->GetCallToken2());

    if (session1 == NULL || session2 == NULL) {
      return TSPF_INVALIDCALLTOKEN;
    }

    Synchronized sync1(session1);
    Synchronized sync2(session2);

    // no need to check for session->IsGoingAway()
    if (!session1->IsGoingAway() && !session2->IsGoingAway())
    {
      if (cServer->GetSessionMgr()->BreakTrombone(session1->cTelephonyChannel,
                                  session2->cTelephonyChannel))
        return TSPF_OK;
/*
        NotifyCallsDisconnected(VUIClientID, pMsg->GetCallToken1(), pMsg->GetCallToken2());
      return true;
*/
      else
        return TSPF_SYSTEMFAILURE;
    }
    else
      return TSPF_INVALIDCALLTOKEN;
/*
      NotifyFailure(VUIClientID, pMsg->GetCallToken(), pMsg->GetSerialNo(), TSPF_INVALIDCALLTOKEN);
    return false;
*/
  }
  void
  TelephonyServiceProvider::CleanupAllResources(void)
  {
    TelephonyChannel *pChannel;
    while ((pChannel = cServer->GetSessionMgr()->GetFirstTelephonyChannel()))
    {
      SessionNode* session = cServer->GetSessionMgr()->FindSessionNode(pChannel);
      // do not check for session->IsGoingAway()
      if (session)
      {
        Synchronized sync(session);
        pChannel->CleanupResources();
        cServer->GetSessionMgr()->DestroySession(pChannel);
      }
    }
  }
  TelephonyChannel *
  TelephonyServiceProvider::FindAvailableChannelForIncomingCall(void)
  {
        std::list<Channel *>::iterator iter = cChannels.begin();
        while ((iter != cChannels.end()) &&
                (((TelephonyChannel *)(*iter))->GetState() !=
                        TelephonyChannel::S_Disconnected &&
                !(((TelephonyChannel *)(*iter))->GetCapabilities() &
                        TelephonyChannel::C_Incoming)))
                iter++;
```

```
        return (iter != cChannels.end()) ? ((TelephonyChannel *)(*iter)) : 0;
}
TelephonyChannel *
TelephonyServiceProvider::FindAvailableChannelForOutgoingCall(void)
{
        //std::list<Channel *>::iterator iter = cChannels.begin();
  for (int count=0; count<2; count++)
  {
    if (cIter == cChannels.end())
    {
      ::Sleep(1000);
      debug("Sleep 1000 msecs before iterating again");
      cIter = cChannels.begin();
    }
    while ((cIter != cChannels.end()) &&
                ((* cIter)->GetState() != Channel::S_Idle ||
                !((TelephonyChannel *)(*cIter))->IsAvailable() ||
      /* ((TelephonyChannel *)(*iter))->GetState() !=
                TelephonyChannel::S_Disconnected ||
                ((TelephonyChannel *)(*iter))->GetStatus() !=
                TelephonyChannel::S_InService ||
                */
                !(((TelephonyChannel *)(*cIter))->GetCapabilities() & TelephonyChannel::C_Outgoing)))
                cIter++;
    if (cIter != cChannels.end())
      break;
  }
  TelephonyChannel *pChan = (cIter != cChannels.end()) ? ((TelephonyChannel *)(*cIter)) : 0;
  if (cIter != cChannels.end())
    cIter++;
  if (cIter == cChannels.end())
    cIter = cChannels.begin();
        return pChan;
}
class  RejectCallHelper : public Thread {
public:
 invui::Call_var  m_call;
 VUISessionImpl*  m_pNode;
 RejectCallHelper(invui::Call_var  call, VUISessionImpl*  pNode)
 {
  m_call = call;
  m_pNode = pNode;
 }

 virtual void run(void)
 {
  gm::string prompt;
  if(ConfigMgr::GetString( MTS_REGISTRY_BASE ".Telephony Service Providers",
        "NoVUIClientPrompt", prompt))
  {
        gm::string alt_tts = "There is no VUI Client. Call rejected.";
        int pos = prompt.find(',');
   if(pos>=0)
        {
         alt_tts = (LPCSTR)prompt.substr(pos+1).c_str();
         prompt.erase(pos);
```

```
        }
    invui::Prompt::audio_abs_s audio;
    audio.prompt = (LPCSTR)prompt.c_str();
    audio.alternative_tts = (LPCSTR)alt_tts.c_str(); // deep copy
        invui::Prompt p;
        p.audio_abs(audio);

        m_pNode->AppendPrompt(p);
    m_pNode->PlayPrompts();
   }
   m_pNode->rejectCall(m_call);
  }
};
CMSMessage::SerialNumberT
TelephonyServiceProvider::NotifyIncomingCall(CallToken& callToken,
        Address& DNIS, Address& ANI, char *UUI, int uuiLength)
{
    TelephonyChannel *pChannel = (TelephonyChannel *)FindChannelForCallToken(callToken);
    // Save for CDR_LOG. Target is DNIS and and CallingNumber is ANI
    VUISessionImpl *pSessionNode = cServer->GetSessionMgr()->CreateSession(pChannel, DNIS, ANI);
    Synchronized sync(pSessionNode);
    // no need to check for session->IsGoingAway()
    pSessionNode->UpdateCallToken();
    IncomingCallNotification ntfy(callToken, DNIS, ANI);
    // DFB_TBD - search through all server nodes if necessary to
    // find one to handle this call.  This must also respect the
    // group id
    //
    invui::Call_var call = new invui::Call();
    try {
      call->session = pSessionNode->duplicateReference();
    } catch (const CORBA::Exception& e) {
      error("NotifyIcomingCall : could not duplicate the session reference ");
      cerr << "TelephonyServiceProvider::NotifyIncomingCall " << e << endl;
    }
    call->id = callToken.GetSequenceNumber();
    diag("[%d] NotifyIncomingCall : finding VUIClient", call->id);
    CallTokenMgr::Register(call->id, callToken);
    invui::VUIClient_var client;
    invui::UUIType uui(256, uuiLength, (unsigned char *)UUI);
        //invui::UUIType uui;
    //uui.length(0);
    const char* dnis = DNIS.GetString();
    const char* ani = ANI.GetString();
        ani = ani == NULL ? "" : ani;
        dnis = dnis == NULL ? "" : dnis;
    MTSServerImpl* mtsserver_impl = SessionMgr::GetSessionMgr()->GetMTSServerImpl();
    while (true) {
      client = mtsserver_impl->GetVUIClient();
      if (client == NULL) {
                        diag("NotifyIncomingCall: No VUIClient, ani=%s, dnis=%s",ani,dnis);
                        (new RejectCallHelper(call,pSessionNode))->start();
        //pSessionNode->rejectCall(call);
        // reject
        return 0;
      }
```

```
    try {
      if (client->handleCall(call, dnis, ani, uui)) {
        break;
      }
    } catch(CORBA::Exception &e) {
      error("NotifyIcomingCall : could not find VUIClient for the call from %s", dnis);
      cerr << "TelephonyServiceProvider::NotifyIncomingCall(2) " << e << ", reference = " << client << endl;
      pSessionNode->rejectCall(call);
      return 0;
    }
  }
  info("[%d] : NotifyIncomingCall , a session started", call->id);
  pSessionNode->SetVUIClient(invui::VUIClient::_duplicate(client));
  ConnectionMgr::addConnectionListener(client, pSessionNode);
  return 0;
}
void
TelephonyServiceProvider::NotifyChannelStatus(TelephonyChannel::ChannelStatus status, const TelephonyChannel&
tChannel)
{
        if (status == TelephonyChannel::S_OutOfService) {
                // decrement
                if (tChannel.GetCapabilities() & TelephonyChannel::C_Outgoing)
                        cTotalNumOutboundChans--;
//              if (!cTotalNumOutboundChans)
//                      Restart();
        }
}
bool
TelephonyServiceProvider::GetOutboundChannelAvailable(void)
{
        return (bool)(FindAvailableChannelForOutgoingCall() != 0);
}
void
TelephonyServiceProvider::NotifyCallBooted(SessionNode *session)
{
        // no need to check for session->IsGoingAway()
        if (session)
        {
                session->cTelephonyChannel->ResetRouting();
                cServer->GetSessionMgr()->DestroySession(session->cTelephonyChannel);
                //session.ReleaseNode();
        }
}
bool
TelephonyServiceProvider::BeginService(void)
{
  // put unused channels in the "Out of Service" state
  int numMChans = cServer->GetMediaServiceProvider()->GetNumChannels();
  int numTChans = cServer->GetTelephonyServiceProvider()->GetNumChannels();
  if (numMChans < numTChans*2)
  {
    int shortage = numTChans*2 - numMChans;
    int numUnusedTChans = shortage % 2 ? shortage/2 + 1 : shortage/2;
    // Set "numUnusedTChans" telephony channels to out-of-service
```

```
        std::list<Channel *>::iterator iter = cChannels.begin();
        while ((iter != cChannels.end()) &&
         numUnusedTChans)
    {
      if ( ((TelephonyChannel *)(*iter))->GetCapabilities() & TelephonyChannel::C_Unused)
      {
                    ((TelephonyChannel *)(*iter))->SetStatus(TelephonyChannel::S_OutOfService);
                    ((TelephonyChannel *)(*iter))-
>SetPhysicalChannelStatus(TelephonyChannel::S_OutOfService);
        numUnusedTChans--;
      }
                iter++;
    }
    iter = cChannels.begin();
        while ((iter != cChannels.end()) &&
         numUnusedTChans)
    {
      ((TelephonyChannel *)(*iter))->SetCapabilities(TelephonyChannel::C_Unused);
                        debug("Marking channel %d unused", ((TelephonyChannel *)(*iter))->GetID());
            ((TelephonyChannel *)(*iter))->SetStatus(TelephonyChannel::S_OutOfService);
                ((TelephonyChannel *)(*iter))->SetPhysicalChannelStatus(TelephonyChannel::S_OutOfService);
      numUnusedTChans--;
                iter++;
    }
  }
  return true;
}
```

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <time.h>
#include <sys/timeb.h>
#include "CMS/CMSClient.h"
//#include "CMS/IPCPipe.h"
#include "CMS/IPCSocket.h"
#include "VPPUtils/CmdLine.h"
#include "VPPUtils/VPPDebug.h"
#include "VPPUtils/GMString"
#define TESTMSGS 1000
int num;
DWORD times[TESTMSGS];
HANDLE ReplyReceivedEvent;
bool
PingFailureMessageHandler(CMSMessage *pMsg, CMSEndpoint *pEndpoint,
        void *userdata)
{
        printf("Server not responding to ping!\n");
        return true;
}
DWORD
GetTimestamp(void)
{
        struct _timeb tb;
        _ftime(&tb);
        struct tm *time = localtime(&tb.time);
        return (DWORD)tb.millitm +
                ((DWORD)time->tm_sec * 1000L) +
                ((DWORD)time->tm_min * 60000L) +
                ((DWORD)time->tm_hour * 3600000L);
}
bool
TimerMessageHandler(CMSMessage *pMsg, CMSEndpoint *pEndpoint,
        void *userdata)
{
        DWORD tc = GetTimestamp();
        times[num++] = tc - pMsg->GetUserdata();
        ::SetEvent(ReplyReceivedEvent);
        printf("Got reply %d at %ld with start %ld, delta %ld\n", num, tc, pMsg->GetUserdata(),
                tc - pMsg->GetUserdata());

        if (num == TESTMSGS)
        {
                DWORD total = 0;
                for (int i = 0; i < TESTMSGS; i++)
                {
//                      printf("times[%d] = %ld, ", i, times[i]);
                        total += times[i];
                }
                DWORD avg = total / TESTMSGS;
                printf("\nTotal of %d messages was %ld ticks\n", TESTMSGS, total);
                printf("\nRound-trip average of %d messages was %ld ticks\n", TESTMSGS, avg);
                fflush(stdout);
```

```
        }
        return true;
}
void
main(int ac, char **av)
{
        char machinename[100];
        gm::string logname = "c:\\testclient.";
        char buf[20];
        itoa(_getpid(), buf, 16);
        logname += buf;
        logname += ".out";
        debug::SetTeeOutput(logname);
        CommandLine cmdline(ac, av);
        gm::string value = cmdline.GetValue("MachineName");
        if (value)
                strcpy(machinename, value);
        else
                strcpy(machinename, ".");
        printf("Attempting to connect to server on %s\n", machinename);
#ifdef USEPIPE
        IPCPipe *pEndpoint = new IPCPipe(machinename, "testpipe");
#else
        IPCSocket *pEndpoint = new IPCSocket(machinename, 1234);
#endif
        CMSClient client(pEndpoint);
        client.AddMessageHandler(0xaaaa, TimerMessageHandler, 0);
        client.AddMessageHandler(C_NTFY_PINGFAILURE, PingFailureMessageHandler, 0);

        if (client.Connect())
        {
                unsigned short id = 0xdfb;
                CMSMessage msg, recvmsg;
                while (client.IsConnected())
                {
                        msg.SetCommandID(id);
                        printf("SPACE = send one msg., C = send 64K msgs.\n");
                        printf("T = time msg, ESC = quit\n");
                        int ch = getch();
                        int nummsgs;
                        bool TimeIt = false;
                        if (ch == ' ')
                                nummsgs = 1;
                        else if (ch == 'C' || ch == 'c')
                                nummsgs = 0xffff;
                        else if (ch == 'T' || ch == 't')
                        {
                                nummsgs = 1;
                                TimeIt = true;
                                msg.SetCommandID(0xaaaa);
                        }
                        else if (ch == 27)
                                break;
                        else
                                nummsgs = 0;
                        while (client.IsConnected() && nummsgs--)
```

```
                {
                        if (TimeIt)
                        {
                                ReplyReceivedEvent = ::CreateEvent(NULL, TRUE, FALSE, NULL);
                                DWORD StartTime = GetTimestamp();
                                num = 0;
                                for (int i = 0; i < TESTMSGS; i++)
                                {
                                        msg.SetUserdata(GetTimestamp());
                                        if (!client.WriteMessage(msg))
                                                printf("Client couldn't write message.\n");
                                        ::WaitForSingleObject(ReplyReceivedEvent, INFINITE);
                                        ::ResetEvent(ReplyReceivedEvent);
                                }
                                DWORD EndTime = GetTimestamp();
                                printf("\nTotal unit time was %ld ticks, average %ld\n",
                                        EndTime - StartTime, ((EndTime - StartTime) / TESTMSGS));
                        }
                        else if (!client.WriteMessage(msg))
                                printf("Client couldn't write message.\n");
                        else
                                printf("%d: Client wrote message ok.\n", msg.GetSerialNo());
                        if (kbhit())
                                break;
                        /*
                        while (client.MessagesWaiting())
                        {
                                CMSMessage *pMsg = client.GetNextMessage();
                                printf("Got message %d (%d)\n", pMsg->GetCommandID(),
                                        pMsg->GetSerialNo());
                                delete pMsg;
                        }
                        */
                }
        }
        client.Disconnect();
    }
}
```

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "CMS/CMSServer.h"
#include "CMS/IPCPipe.h"
#include "CMS/IPCSocket.h"
#include "VPPUtils/VPPDebug.h"
#include "VPPUtils/CmdLine.h"
#include "VPPUtils/GMString"
//#define USEPIPE
bool
SpecialMessageHandler(CMSMessage *pMsg, CMSEndpoint *pEndpoint, void *userdata)
{
        static int num = 1;
        printf("%d: Got special message!\n", num++);
        return true;
}
bool
TransactionMessageHandler(CMSMessage *pMsg, CMSEndpoint *pEndpoint, void *userdata)
{
//      printf("Got transaction message with start time %ld!\n", pMsg->GetUserdata());
        pEndpoint->WriteMessage(*pMsg);
        return true;
}
bool
PingFailureMessageHandler(CMSMessage *pMsg, CMSEndpoint *pEndpoint,
        void *userdata)
{
        printf("Client not responding to ping!\n");
        return true;
}
void
main(int ac, char **av)
{
        CommandLine cmdline(ac, av);
        debug::SetTeeOutput("c:\\testserver.out");
#ifdef USEPIPE
        IPCEndpoint *pEndpoint = new IPCPipe(".", "testpipe");
#else
        IPCEndpoint *pEndpoint = new IPCSocket("", 1234);
#endif
        CMSServer server(pEndpoint);
        server.AddMessageHandler(0xdfb, SpecialMessageHandler, 0);
        server.AddMessageHandler(0xaaaa, TransactionMessageHandler, 0);
        server.AddMessageHandler(C_NTFY_PINGFAILURE, PingFailureMessageHandler, 0);
        if (!server.Listen())
                printf("Error- Listen failed!\n");
        unsigned long stdhdl = (unsigned long)::GetStdHandle(STD_INPUT_HANDLE);
        while (true)
        {
                int ret = server.WaitForClientConnect(1, &stdhdl);
                if (ret == 2)
                {
                        INPUT_RECORD input_event;
                        DWORD num_read;
                        ::PeekConsoleInput((HANDLE)stdhdl, &input_event, 1, &num_read);
```

```
                    if (num_read > 0)
                    {
                        if (input_event.EventType != KEY_EVENT ||
                            !input_event.Event.KeyEvent.bKeyDown ||
                            !input_event.Event.KeyEvent.uChar.AsciiChar)
                            ::ReadConsoleInput((HANDLE)stdhdl, &input_event, 1, &num_read);
                        else
                        {
                            // A key was pressed
                            //
                            char ch;
                            ::ReadFile((HANDLE)stdhdl, &ch, 1, &num_read, 0);
                            if (ch == 13 || ch == 27)
                                break;
                        }
                    }
                }
            }
        }
```

```
/*
 * Thread.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.7 $
 * $Date: 2000/12/07 03:22:19 $
 */
static const char* id="$Id: Thread.cpp,v 1.7 2000/12/07 03:22:19 Mitsuru Oshima Exp $";
#include <assert.h>
#include <windows.h>
#include <process.h>
#include <stdio.h>
#include "Thread.h"
#include "UtilsLog.h"
void __run(void* v) {
   Thread *pThis = (Thread *)v;
   pThis->run();
   if (pThis->isAutodelete()) {
      UtilsLog::debug("deleting thread id=%ld", pThis);
      delete pThis;
   }
}
/*
 * This may be buggy...
 */
void
Thread::join() {
   HANDLE h = t_handle;
   if (h == NULL) {
      UtilsLog::error("No Thread in Thread::join");
      return;
   }
   ::WaitForSingleObject((HANDLE)h, INFINITE);
}
/*
 */
void
Thread::setPriority(int p) {
   HANDLE h = t_handle;
   priority = p;
   if (h != NULL) {
      ::SetThreadPriority(h, priority);
   }
}
void
ThreadR::run() {
   if (runnable != NULL) {
      runnable->run();
   }
}
void
```

```
Thread::start() {
   HANDLE h = t_handle = (HANDLE)::_beginthread(__run, 0, this);
   setPriority(priority);
}
```

```
/*
 * Timer.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.12 $
 * $Date: 2000/12/07 03:22:19 $
 */
static const char* id="$Id: Timer.cpp,v 1.12 2000/12/07 03:22:19 Mitsuru Oshima Exp $";
#include "Timer.h"
#include "Synchronized.h"
#include "Thread.h"
#include "UtilsLog.h"
const int OFFSET = 100; // within 0.1 seconds
static Timer::TimerID timer_id = 0;
/**
 * Global Timer
 */
Timer GTimer::timer = Timer();
Timer::TimerID
GTimer::add(TimerEvent* event) {
  return timer.add(event);
}
bool
GTimer::remove(Timer::TimerID id) {
  return timer.remove(id);
}
void
GTimer::start(int n, int p) {
  for (int i=0; i<n; i++) {
    Thread* t = new ThreadT<Timer>(&timer, Timer::run);
    t->setPriority(p);
    t->start();
  }
}
void
GTimer::stop() {
  timer.stop();
}
/**
 * Timer Queue
 */
Timer::Timer() : _running(true), bsize(5), elementCount(1), numOfThreads(0) {
  elementData = new TE_PTR[5];
  // [0] is not used
  elementData[0] = NULL;
}
Timer::~Timer() {
  stop();
  delete [] elementData;
}
void
```

```
Timer::run() {
  {
    Synchronized sync(this);
    numOfThreads ++; // probably we may not need this.
  }
  while(_running) {
    TimerEvent* e = dequeue();
    if (e == NULL) {
      // skip
    } else if (e->isRepeating()) {
      add(e);
      e->run();
    } else {
      UtilsLog::info("executing event ");
      e->run();
      delete e;
    }
  }
  Synchronized sync(this);
  numOfThreads --;
  if (numOfThreads == 0) {
    sync.notifyAll();
  }
}
void
Timer::start(int n, int p) {
  for (int i=0; i<n; i++) {
    Thread* t = new ThreadT<Timer>(this, Timer::run);
    t->setPriority(p);
    t->start();
  }
}
Timer::TimerID
Timer::add(TimerEvent* event) {
  Synchronized sync(this);
  Timer::TimerID id = timer_id ++;
  event->setID(id);
  // UtilsLog::info("adding event which wakes up at %ld", event->when);
  int pos = bsearch(event);
  insertElementAt(event, pos);
  if (pos == (elementCount -1)) {
    sync.notifyAll();
  }
  return id;
}

unsigned long
Timer::valueAt(int x) {
  return elementData[x]->when;
}
int
Timer::bsearch(TimerEvent* event) {
  int l = 1, r = (elementCount-1), x = 1;
  unsigned long w = event->when;
  while (r >= l) {
    x = (l + r)/2;
```

```
        unsigned long c = valueAt(x);
        if (w == c) {
          // append at the end
          while (x>1 && w == valueAt(x)) {
            x--;
          }
          if (valueAt(x) > w) {
            x ++;
          }
          return x;
        }
        if ( w > c) {
          r = x-1;
          if (r < l) {
            while (x>1 && w >= valueAt(x)) {
              x --;
            }
            if (valueAt(x) > w) {
              x ++;
            }
          }
        } else {
          l = x+1;
          if (r < l) {
            while (x<elementCount && w < valueAt(x)) {
              x ++;
            }
          }
        }
      }
      return x;
    }
    class IdFilter : public Timer::Filter {
    private:
      Timer::TimerID _id;
      bool _stop;
    public:
      IdFilter(Timer::TimerID id) : _id(id), _stop(false) {
      }
      virtual bool accept(TimerEvent* ev) {
        return (_stop = (_id == ev->getID()));
      };
      virtual bool stop() {
        return _stop;
      };
    };
    bool
    Timer::remove(Timer::TimerID id) {
      IdFilter flt(id);
      return remove(flt) > 0;
    }
    int
    Timer::remove(Filter& flt) {
      Synchronized sync(this);
      int i = 1;
      int numOfRemoved = 0;
```

```
    for (; i<elementCount; i++) {
      TimerEvent* ev = elementData[i];

      if (flt.accept(ev)) {
        numOfRemoved ++;
        removeElementAt(i);
        delete ev;
        if (flt.stop()) {
          break;
        }
      }
    }
    // if the top most event has been removed.
    // this is not elementCount - 1, because it's been decreased
    if (i == elementCount) {
      sync.notifyAll();
    }
    return numOfRemoved;
}
void
Timer::stop() {
    Synchronized sync(this);
    if (_running == false) {
      return;
    }
    _running = false;
    sync.notifyAll();
    while(numOfThreads>0) {
      sync.wait();
    }
}
TimerEvent*
Timer::dequeue() {
    Synchronized sync(this);
    while(_running) {
      if (elementCount == 1) { // no data
        sync.wait();
      } else {
        TimerEvent* e = lastElement();
        unsigned long current = ::GetTickCount();
        unsigned long diff = e->when - current;
        if (e->when < current || diff < OFFSET) {
          removeElementAt(elementCount -1);
          return e;
        } else {
          sync.wait(diff);
        }
      }
    }
    return NULL;
}
void
Timer::arraycopy(TE_PTR* src, int src_pos, TE_PTR* to, int to_pos, int length) {
    src = & (src[src_pos]);
    to = & (to[to_pos]);
    if (src == to) {
```

```
        return;
    } else if (src < to) {
        for (int i=length-1; i>=0; i--) {
            to[i] = src[i];
        }
    } else {
        memcpy(to, src, length * sizeof(TE_PTR));
    }
};
void
Timer::insertElementAt(TimerEvent* ev, int at) {
    int newcount = elementCount + 1;
    if (at >= newcount) {
        UtilsLog::error("Internal Error in Timer::insertElementAt");
        return;
/*
        throw new Error("Internal Error");
*/
    }
    if (newcount > bsize) {
        int oldCapacity = bsize;
        TE_PTR* oldData = elementData;
        bsize = oldCapacity * 2;
        elementData = new TE_PTR[bsize];
        arraycopy(oldData, 0, elementData, 0, elementCount);
        delete [] oldData;
    }
    arraycopy(elementData, at, elementData, at + 1,
            elementCount - at);
    elementData[at] = ev;
    elementCount++;
}
void
Timer::removeElementAt(int at) {
    if (at >= elementCount) {
        UtilsLog::error("Internal Error in Timer::removeElementAt");
        return;
        // throw new Error("Internal Error");
    }
    else if (at < 0) {
        UtilsLog::error("Internal Error in Timer::removeElementAt");
        return;
        // throw new Error("Internal Error");
    }
    int j = elementCount - at - 1;
    if (j > 0) {
        arraycopy(elementData, at + 1, elementData, at, j);
    }
    elementCount--;
    elementData[elementCount] = NULL;
}
TimerEvent*
Timer::lastElement() {
    if (elementCount == 0) {
        UtilsLog::error("Internal Error in Timer::lastElement");
        return NULL;
```

```
/*
      throw new Error("Internal Error");
*/
    }
    return elementData[elementCount - 1];
}
/*
class EventEnumerator implements Enumeration {
      private int i = 1;
      private Object next = null;
      private Filter filter = null;
      EventEnumerator(Filter flt) {
        filter = flt;
      }

      public bool hasMoreElements() {
        if (next != null) {
          System.out.println("yes");
          return true;
        }
        System.out.println("checking " + i);
        int size = size();
        for (; i<size; i++) {
          TimerEvent ev = (TimerEvent)elementAt(i);
          if (ev intasnceof BackupEvent) {
            continue;
          }

          if (filter.accept(ev)) {
            next = ev;
            if (filter.stop()) {
              i = size;
            } else {
              i++;
            }
            break;
          }
        }
        return next != null;
      }
      public Object nextElement() {
        if (next == null) {
          hasMoreElements();
        }
        Object n = next;
        next = null;
        return n;
      }
    }
    public Enumeration elements(Filter flt) {
      return new EventEnumerator(flt);
    }
    static public interface Filter {
      public bool accept(TimerEvent ev);
      public bool stop();
    };
```

```
}
*/
```

Downloaded from http://www.dbebooks.com

```
/*
 * TTSChannel.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.19 $
 * $Date: 2000/12/07 03:19:54 $
 */
static const char* id="$Id: TTSChannel.cpp,v 1.19 2000/12/07 03:19:54 Mitsuru Oshima Exp $";
#include <assert.h>
#include <windows.h>
#include <process.h>
#include <stdio.h>
#include "VPPUtils/Log.h"
#include "VPPUtils/Synchronize.h"
#include "TTSChannel.h"
#include "SessionMgr.h"
static void _TTSChannelThread(void *);
TTSChannel::TTSChannel(ServiceProvider *pServiceProvider,
        const char *logType, int id, bool RequiresSettlement)
  : Channel(pServiceProvider, Channel::CT_TTS, logType, id, RequiresSettlement),
  m_State(TTS_Invalid),
  m_pString(NULL)
{
}
bool
TTSChannel::Shutdown(void)
{
        return true;
}
TTSChannel::~TTSChannel()
{
        if(m_pString)
        {
                delete [] m_pString;
        }
}
bool TTSChannel::InitializeSynthesis(const char *TTSstring)
{
        if (!TTSstring || !strlen(TTSstring)) {
                warning("TTS is requested for an empty or null string.");
                return false;
        }
        info("Start TTS for string");
        diag("String: %s", TTSstring);
        Synchronized sync(m_StateMonitor);
        if (m_State != TTS_Idle)
        {
                info("Waiting for previous TTS to stop");
                sync.wait(1000); // Wait a second or till idle
                if (m_State != TTS_Idle)
                {
```

```
                warning("Trying to stop previous TTS");
                StopTTS();
                sync.wait(3000); // Wait a couple seconds or till idle
            }
        }
        if (m_State == TTS_Idle)
        {
            if(m_pString)
            {
                delete [] m_pString;
            }
            m_pString = new char[strlen(TTSstring)+1];
            strcpy(m_pString, TTSstring);

            m_State = TTS_Initialized;
            return(StartTTS());
        }
        else
        {
            error("Couldn't stop previous TTS");
            return false;
        }
}
int TTSChannel::GetNextSamples(char *buffer, unsigned long bufferSize)
{
        int retLen = bufferSize;
        if(m_State == TTS_Initialized)
        {
            int dataSize;
            switch(SynthesisStatus())
            {
            case TTSSynthStatus_NotReady:
                memset(buffer, '\xFF', bufferSize);
                debug("No bytes available -- returning silence");
                break;
            case TTSSynthStatus_Ready:
                dataSize = GetNextBuffer(buffer, bufferSize);
                if(dataSize < bufferSize)
                {
                    memset(buffer+dataSize, '\xFF', bufferSize - dataSize);
                    debug("%d bytes are available -- returning silence", dataSize);
                }
                break;
            case TTSSynthStatus_EOD:
                retLen = 0;
                break;
            default:
                retLen = 0;
                break;
            }
        }
        else
        {
            error("GetNextSamples(), state is not TTS_Initialized");
            retLen = 0;
        }
```

```
        return retLen;
}
void TTSChannel::Release(void)
{
        diag("Releasing TTS");
        {
                Synchronized sync(m_StateMonitor);
                m_State = TTS_Stop;
        }
        StopTTS();
}
// virtual
void
TTSChannel::NotifyBoundToSession(SessionNode *pSessionNode)
{
        Channel::NotifyBoundToSession(pSessionNode);
        if (!pSessionNode)
                NotifySettled();
}
void
TTSChannel::NotifySettled(void)
{
        info("TTSChannel::NotifySettled, m_State = %d", m_State);
        Synchronized sync(SessionMgr::GetSessionMgr());
  Channel::NotifySettled();
        sync.notifyAll();
}
```

```
/*
 * TTSMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:59 $
 */
static const char* id="$Id: TTSMsg.cpp,v 1.3 2000/12/07 03:21:59 Mitsuru Oshima Exp $";
#include "TTSMsg.h"
// virtual
void
TTSMessage::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        InsertString(cText);
        InsertString(cProductName);
        InsertUshort(cLangID);
        InsertUshort(cSpeed);
        InsertUshort(cPitch);
}
// virtual
void TTSMessage::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        ExtractString(cText);
        ExtractString(cProductName);
        cLangID = (GMLANGID)ExtractUshort();
        cSpeed = ExtractUshort();
        cPitch = ExtractUshort();
}
```

```
/*
 * TTSSegment.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.6 $
 * $Date: 2000/12/07 03:19:55 $
 */
static const char* id="$Id: TTSSegment.cpp,v 1.6 2000/12/07 03:19:55 Mitsuru Oshima Exp $";
#include "TTSSegment.h"
#include "MediaChannel.h"
// virtual
TTSSegment::~TTSSegment(void)
{
}
void
TTSSegment::Append(const gm::string& ttsText)
{
        if (cTTSText[cTTSText.length() - 1] != ' ' &&
                ttsText.c_str()[0] != ' ')
                cTTSText += " ";
        cTTSText += ttsText;
}
// virtual
bool
TTSSegment::Play(MediaChannel *pMediaChannel,
          CallToken& ct)
{
        return (bool)(pMediaChannel->DoPlayTTS(ct, cTTSText) == MSPF_OK);
}
```

```
/*
 * TTSServiceProvider.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.5 $
 * $Date: 2000/12/07 03:19:55 $
 */
static const char* id="$Id: TTSServiceProvider.cpp,v 1.5 2000/12/07 03:19:55 Mitsuru Oshima Exp $";
#include "TTSServiceProvider.h"
#include "Channel.h"
// virtual
TTSServiceProvider::~TTSServiceProvider(void)
{
  Shutdown();
}
TTSChannel *
TTSServiceProvider::FindAvailableChannel(void)
{
        std::list<Channel *>& ttschannels = GetChannels();
        std::list<Channel *>::iterator ttschiter = ttschannels.begin();
        while (ttschiter != ttschannels.end())
        {
                if ((*ttschiter)->GetState() == Channel::S_Idle)
                {
                        return (TTSChannel *)(*ttschiter);
                }
                ttschiter++;
        }
        return 0;
}
```

```
/*
 * TwoCallTokensMsgBase.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.4 $
 * $Date: 2000/12/07 03:21:59 $
 */
static const char* id="$Id: TwoCallTokensMsgBase.cpp,v 1.4 2000/12/07 03:21:59 Mitsuru Oshima Exp $";
#include "TwoCallTokensMsgBase.h"
// virtual
void
TwoCallTokensMessageBase::Serialize(void)
{
        OneCallTokenMessageBase::Serialize();
        cCallToken2.Serialize(this);
}
// virtual
void
TwoCallTokensMessageBase::Deserialize(void)
{
        OneCallTokenMessageBase::Deserialize();
        cCallToken2.Deserialize(this);
}
```

```
/*
 * UtilsLog.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.2 $
 * $Date: 2000/12/07 03:22:20 $
 */
static const char* id="$Id: UtilsLog.cpp,v 1.2 2000/12/07 03:22:20 Mitsuru Oshima Exp $";
#include "UtilsLog.h"
UtilsLog UtilsLog::log;
```

Document 2 Page 20

```
/*
 * VPPDebug.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.16 $
 * $Date: 2000/12/07 03:22:20 $
 */
static const char* id="$Id: VPPDebug.cpp,v 1.16 2000/12/07 03:22:20 Mitsuru Oshima Exp $";
#undef UTILLIB
#include <windows.h>
#include <sys/timeb.h>
#include <time.h>
#include <stdarg.h>
#include <limits.h>
#include "VPPDebug.h"
#include "Synchronize.h"
#include "ConfigMgr.h"
#ifdef UTILLIB
#include "Utillib/Log.h"
#endif
#define DEFAULT_REPARSE_TIME ((unsigned long)60000L)
FILE *debug::cOutput = stderr;
FILE *debug::cTeeOutput = NULL;
CriticalSection debug::cCriticalSection;
std::list<debug::DebugNode *> debug::cGroups;
bool debug::cAtNewline = true;
bool debug::cInitialized = false;
bool debug::cAllPresent = false;
bool debug::cAllSquelched = false;
unsigned long debug::cLastCheck = 0;
unsigned long debug::cReparseTime = DEFAULT_REPARSE_TIME;
void
debug::Init(void)
{
        cInitialized = true;
#ifdef UTILLIB
        Log::setOutputLevel(Log::PHONE, Log::LOG_DEBUG);
#endif
        ParseDebugGroups();
        cLastCheck = ::GetTickCount();
}
void
debug::ParseDebugGroups(void)
{
        gm::string regstr;
        char *str;
        if (ConfigMgr::GetString(MTS_REGISTRY_BASE, "DebugReparseTime", regstr))
                cReparseTime = ::atol(regstr);
        if (ConfigMgr::GetString(MTS_REGISTRY_BASE, "DebugGroups", regstr) &&
                (str = strdup(regstr)))
        {
```

```
                std::list<DebugNode *>::iterator iter = cGroups.begin();
                while (iter != cGroups.end())
                {
                        delete *iter;
                        iter++;
                }
                cGroups.clear();
                char *substr = 0;
                while ((substr = strtok(substr ? 0 : str, ", \t;")))
                {
                        if (strcmp(substr, "All"))
                        {
                                DebugNode *pNode = new DebugNode;
                                pNode->name = strdup(substr);
                                pNode->squelched = false;
                                cGroups.push_back(pNode);
                        }
                        else
                        {
                                cAllPresent = true;
                                cAllSquelched = false;
                        }
                }
                free(str);
        }
}
bool
debug::IsEnabled(const char *group)
{
        if (!cInitialized)
                Init();
        if (!group)
                return true;
        std::list<DebugNode *>::iterator iter = cGroups.begin();
        while (iter != cGroups.end())
        {
                if (!strcmp((*iter)->name, group) && !((*iter)->squelched))
                        return !cAllSquelched;
                iter++;
        }
        return cAllPresent & !cAllSquelched;
}
void
debug::SetOutput(FILE *output)
{
        Synchronize sync(cCriticalSection);
        cOutput = output;
}
void
debug::SetOutput(const char *filename, bool overwrite)
{
        SetOutput(fopen(filename, overwrite ? "wt" : "a+t"));
        if (cOutput)
                printf(0, "Output changed to %s\n", filename);
}
FILE *
```

```
debug::GetOutput(void)
{
	Synchronize sync(cCriticalSection);
	return cOutput;
}
void
debug::SetTeeOutput(FILE *teeoutput)
{
	Synchronize sync(cCriticalSection);
	cTeeOutput = teeoutput;
}
void
debug::SetTeeOutput(const char *filename, bool overwrite)
{
	SetTeeOutput(fopen(filename, overwrite ? "wt" : "a+t"));
	if (cTeeOutput)
		printf(0, "Tee output changed to %s\n", filename);
}
FILE *
debug::GetTeeOutput(void)
{
	Synchronize sync(cCriticalSection);
	return cTeeOutput;
}
void
debug::Squelch(const char *group, bool squelch)
{
	if (!cInitialized)
		Init();
	if (!strcmp(group, "All"))
		cAllSquelched = squelch;
	else
	{
		std::list<DebugNode *>::iterator iter = cGroups.begin();
		while (iter != cGroups.end())
		{
			if (!strcmp((*iter)->name, group))
			{
				(*iter)->squelched = squelch;
				return;
			}
			iter++;
		}
	}
}
void
debug::PrintHeader(FILE *fp)
{
	struct _timeb tb;
	_ftime(&tb);
	struct tm *time = localtime(&tb.time);
	::fprintf(fp, "%.2d.%.2d|%.2d:%.2d:%.2d.%.3d: ", time->tm_mon + 1,
		time->tm_mday, time->tm_hour, time->tm_min, time->tm_sec,
		tb.millitm);
}
int
```

```
debug::doprint(const char *group, const char *fmt, const va_list vl, bool console)
{
	Synchronize sync(cCriticalSection);
	unsigned long tick = ::GetTickCount();
	if (tick - cLastCheck >= cReparseTime)
	{
		ParseDebugGroups();
		cLastCheck = tick;
	}

	if (!group || IsEnabled(group))
	{
		if (cAtNewline)
		{
			if (console)
				PrintHeader(stdout);
			else
			{
#ifndef UTILLIB
				PrintHeader(cOutput);
#endif
				if (cTeeOutput)
					PrintHeader(cTeeOutput);
			}
		}
		int ret = 0;
		if (console)
		{
			ret = vprintf(fmt, vl);
		}
		else
		{
#ifndef UTILLIB
			ret = vfprintf(cOutput, fmt, vl);
			fflush(cOutput);
#else
			Log::vlog(Log::LOG_DEBUG, Log::PHONE, fmt, vl);
			ret = 1;
#endif
			if (cTeeOutput)
			{
				vfprintf(cTeeOutput, fmt, vl);
				fflush(cTeeOutput);
			}
		}
		cAtNewline = (fmt[strlen(fmt) - 1] == 0xa);
		return ret;
	}
	return 0;
}
int
debug::printf(const char *group, const char *fmt, ...)
{
	va_list vl;
	va_start(vl, fmt);
	return doprint(group, fmt, vl, false);
```

```
}
int
debug::console(const char *fmt, ...)
{
        va_list vl;
        va_start(vl, fmt);
        return doprint(0, fmt, vl, true);
}
int
debug::printfandconsole(const char *group, const char *fmt, ...)
{
        va_list vl;
        va_start(vl, fmt);
        doprint(0, fmt, vl, true);
        return doprint(group, fmt, vl, false);
}
int
debug::vlprintf(const char *group, const char *fmt, const va_list vl)
{
        return doprint(group, fmt, vl, false);
}
```

```
/*
 * VPPMsg.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.3 $
 * $Date: 2000/12/07 03:21:59 $
 */
static const char* id="$Id: VPPMsg.cpp,v 1.3 2000/12/07 03:21:59 Mitsuru Oshima Exp $";
#include "VPPMsg.h"
#if 0
#include "VMachineID.h"
// virtual
VirtualMachineID&
VPPMessage::GetMTSVirtualMachineID(void)
{
        static VirtualMachineID InvalidVirtualMachineID;
        return InvalidVirtualMachineID;
}
#endif
```

```
/*
 * VPPMsgMgr.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.23 $
 * $Date: 2000/12/07 03:21:59 $
 */
static const char* id="$Id: VPPMsgMgr.cpp,v 1.23 2000/12/07 03:21:59 Mitsuru Oshima Exp $";
#include "VPPMsgMgr.h"
#include "VPPUtils/VPPDebug.h"
#include "VPPMsg/VPPMsg.h"
#if 0
#include "RegVUIMsg.h"
#endif
#include "PlaceCallMsg.h"
#include "DropCallMsg.h"
#include "ConnectCallsMsg.h"
#include "PlaceCallOnHoldMsg.h"
#include "AcceptCallMsg.h"
#include "RejectCallMsg.h"
#if 0
#include "RegVUIBrokerMsg.h"
#include "WrappedMsg.h"
#endif
#include "CancelRequestMsg.h"
#include "DisconnectCallsMsg.h"
#include "PlayMsg.h"
#include "StopPlayMsg.h"
#include "RecordMsg.h"
#include "StopRecordMsg.h"
#include "RemoveRecordingMsg.h"
#include "CallProgressNtfy.h"
#include "CallDroppedNtfy.h"
#include "CallsConnectedNtfy.h"
#include "CallPlacedOnHoldNtfy.h"
#include "IncomingCallNtfy.h"
#if 0
#include "AssignedClientIDNtfy.h"
#endif
#include "CallConnectedNtfy.h"
#include "RequestCancelledNtfy.h"
#include "SITDetectedNtfy.h"
#include "CNGDetectedNtfy.h"
#include "CallsDisconnectedNtfy.h"
#include "PlayingNtfy.h"
#include "PlayDoneNtfy.h"
#include "StoppedPlayingNtfy.h"
#include "RecordingNtfy.h"
#include "StoppedRecordingNtfy.h"
#include "FailureNtfy.h"
#include "HandoffRecordingMsg.h"
```

```
#include "RecordingHandedOffNtfy.h"
#include "TTSMsg.h"
#include "TTSPlayingNtfy.h"
#include "TTSDoneNtfy.h"
#include "CreateCompositePromptMsg.h"
#include "AppendCompositePromptpromptMsg.h"
#include "AppendCompositePromptSilenceMsg.h"
#include "AppendCompositePromptTTSMsg.h"
#include "EndCompositePromptMsg.h"
#include "PlayCompositePromptMsg.h"
#include "CompositePromptCreatedNtfy.h"
#include "CompositePromptSegmentAppendedNtfy.h"
#include "CompositePromptEndedNtfy.h"
#include "CompositePromptPlayingNtfy.h"
#include "CompositePromptSegmentPlayingNtfy.h"
#include "CompositePromptSegmentPlayDoneNtfy.h"
#include "CompositePromptPlayDoneNtfy.h"
#include "IncomingCallRevokedNtfy.h"
#include "StartRecognitionMsg.h"
#include "StopRecognitionMsg.h"
#include "RecognitionDoneNtfy.h"
#include "RecognizeStringMsg.h"
#include "SetPropertyMsg.h"
#include "GetPropertyMsg.h"
#include "PropertySetNtfy.h"
#include "PropertyGottenNtfy.h"
#include "PropertyNotSetNtfy.h"
#include "PropertyNotGottenNtfy.h"
#include "BeginSpeechNtfy.h"
#include "EndSpeechNtfy.h"
#include "DTMFNtfy.h"
#include "CallAcceptedNtfy.h"
#include "CallRejectedNtfy.h"
#include "RecognitionStartedNtfy.h"
#include "CDRInfoNtfy.h"
#if 0
#include "ProcessShutdownCompleteNtfy.h"
#endif
#include "LogEnterStateMsg.h"
#include "LogItemMsg.h"
#include "LogLeaveStateMsg.h"
#include "CompositeMsg.h"
#include "T2SSPMsg.h"
#include "CreateDynamicGrammarMsg.h"
#include "DeleteDynamicGrammarMsg.h"
#include "AddPhraseToDynamicGrammarMsg.h"
#include "RemovePhraseFromDynamicGrammarMsg.h"
#include "QueryDynamicGrammarContentsMsg.h"
#include "InsertDynamicGrammarMsg.h"
#include "CreateDynamicGrammarResultNtfy.h"
#include "DeleteDynamicGrammarResultNtfy.h"
#include "AddPhraseToDynamicGrammarResultNtfy.h"
#include "RemovePhraseFromDynamicGrammarResultNtfy.h"
#include "QueryDynamicGrammarContentsResultNtfy.h"
#include "InsertDynamicGrammarResultNtfy.h"
```

```
#define RETURNNEW(x) {CMSMessage *ptr = new x; debug::printf("VPPMsgMgr", "Allocated %s at %p\n", #x,
ptr); \
        return ptr;}
CMSMessage *
VPPMsgMgr::AllocateVPPSubclassMessage(CMSMessage *pMsg)
{
        debug::printf("VPPMsgMgr", "AllocateVPPSubclassMessage(%p), S/N %d, CmdId %d\n",
                pMsg, pMsg->GetSerialNo(), pMsg->GetCommandID());
        switch (pMsg->GetCommandID())
        {
#if 0
  case M_CMD_REGISTERVUI:
                        RETURNNEW(RegisterVUIMessage(pMsg));
                case M_CMD_UNREGISTERVUI:
                        break;
#endif
                case M_CMD_PLACECALL:
                        RETURNNEW(PlaceCallMessage(pMsg));
                case M_CMD_DROPCALL:
                        RETURNNEW(DropCallMessage(pMsg));
                case M_CMD_CONNECTCALLS:
                        RETURNNEW(ConnectCallsMessage(pMsg));
                case M_CMD_LAZYSESSIONSHUTDOWN:
                case M_CMD_IMMEDIATESESSIONSHUTDOWN:
                case M_CMD_PROCESSSHUTDOWN:
                        break;
                case M_CMD_PLACECALLONHOLD:
                        RETURNNEW(PlaceCallOnHoldMessage(pMsg));
                case M_CMD_ACCEPTCALL:
                        RETURNNEW(AcceptCallMessage(pMsg));
                case M_CMD_REJECTCALL:
                        RETURNNEW(RejectCallMessage(pMsg));
                case M_CMD_REGISTERVUIBROKER:
                        RETURNNEW(RegisterVUIBrokerMessage(pMsg));
                case M_CMD_UNREGISTERVUIBROKER:
                        break;

                case M_CMD_WRAPPEDMSG:
                        RETURNNEW(WrappedMessage(pMsg));
                case M_CMD_CANCELREQUEST:
                        RETURNNEW(CancelRequestMessage(pMsg));
                case M_CMD_DISCONNECTCALLS:
                        RETURNNEW(DisconnectCallsMessage(pMsg));
                case M_CMD_PLAY:
                        RETURNNEW(PlayMessage(pMsg));
                case M_CMD_STOPPLAY:
                        RETURNNEW(StopPlayMessage(pMsg));
                case M_CMD_RECORD:
                        RETURNNEW(RecordMessage(pMsg));
                case M_CMD_STOPRECORD:
                        RETURNNEW(StopRecordMessage(pMsg));
                case M_CMD_REMOVERECORDING:
                        RETURNNEW(RemoveRecordingMessage(pMsg));
                case M_CMD_HANDOFFRECORDING:
                        RETURNNEW(HandOffRecordingMessage(pMsg));
                case M_CMD_TTS:
```

```
        RETURNNEW(TTSMessage(pMsg));
case M_CMD_CREATECOMPOSITEPROMPT:
        RETURNNEW(CreateCompositePromptMessage(pMsg));

case M_CMD_APPENDCOMPOSITEPROMPTPROMPT:
        RETURNNEW(AppendCompositePromptPromptMessage(pMsg));

case M_CMD_APPENDCOMPOSITEPROMPTSILENCE:
        RETURNNEW(AppendCompositePromptSilenceMessage(pMsg));

case M_CMD_APPENDCOMPOSITEPROMPTTIME:
case M_CMD_APPENDCOMPOSITEPROMPTDATE:
case M_CMD_APPENDCOMPOSITEPROMPTNUMBER:
case M_CMD_APPENDCOMPOSITEPROMPTPHONENUMBER:
        break;

case M_CMD_APPENDCOMPOSITEPROMPTTTS:
        RETURNNEW(AppendCompositePromptTTSMessage(pMsg));

case M_CMD_ENDCOMPOSITEPROMPT:
        RETURNNEW(EndCompositePromptMessage(pMsg));

case M_CMD_PLAYCOMPOSITEPROMPT:
        RETURNNEW(PlayCompositePromptMessage(pMsg));
case M_CMD_STARTRECOGNITION:
        RETURNNEW(StartRecognitionMessage(pMsg));
case M_CMD_STOPRECOGNITION:
        RETURNNEW(StopRecognitionMessage(pMsg));
case M_CMD_RECOGNIZESTRING:
        RETURNNEW(RecognizeStringMessage(pMsg));
case M_CMD_SETPROPERTY:
        RETURNNEW(SetPropertyMessage(pMsg));
case M_CMD_GETPROPERTY:
        RETURNNEW(GetPropertyMessage(pMsg));
case M_CMD_LOGENTERSTATE:
        RETURNNEW(LogEnterStateMessage(pMsg));
case M_CMD_LOGITEM:
        RETURNNEW(LogItemMessage(pMsg));
case M_CMD_LOGLEAVESTATE:
        RETURNNEW(LogLeaveStateMessage(pMsg));
case M_CMD_COMPOSITEMSG:
        RETURNNEW(CompositeMessage(pMsg));
case M_CMD_CREATEDYNAMICGRAMMAR:
        RETURNNEW(CreateDynamicGrammarMessage(pMsg));
case M_CMD_DELETEDYNAMICGRAMMAR:
        RETURNNEW(DeleteDynamicGrammarMessage(pMsg));
case M_CMD_ADDPHRASETODYNAMICGRAMMAR:
        RETURNNEW(AddPhraseToDynamicGrammarMessage(pMsg));
case M_CMD_REMOVEPHRASEFROMDYNAMICGRAMMAR:
        RETURNNEW(RemovePhraseFromDynamicGrammarMessage(pMsg));
case M_CMD_QUERYDYNAMICGRAMMARCONTENTS:
        RETURNNEW(QueryDynamicGrammarContentsMessage(pMsg));
case M_CMD_INSERTDYNAMICGRAMMAR:
        RETURNNEW(InsertDynamicGrammarMessage(pMsg));
case M_RPLY_REGISTERVUI:
        RETURNNEW(RegisterVUIReply(pMsg));
```

```
case M_RPLY_REGISTERVUIBROKER:
        RETURNNEW(RegisterVUIBrokerReply(pMsg));
case M_NTFY_SUSPENDED:
case M_NTFY_SESSIONSHUTDOWNCOMPLETE:
        break;
case M_NTFY_PROCESSSHUTDOWNCOMPLETE:
        RETURNNEW(ProcessShutdownCompleteNotification(pMsg));
        break;
case M_NTFY_CALLPROGRESS:
        RETURNNEW(CallProgressNotification(pMsg));
case M_NTFY_CALLDROPPED:
        RETURNNEW(CallDroppedNotification(pMsg));
case M_NTFY_CALLSCONNECTED:
        RETURNNEW(CallsConnectedNotification(pMsg));
case M_NTFY_CALLPLACEDONHOLD:
        RETURNNEW(CallPlacedOnHoldNotification(pMsg));
case M_NTFY_INCOMINGCALL:
        RETURNNEW(IncomingCallNotification(pMsg));
case M_NTFY_ASSIGNEDCLIENTID:
        RETURNNEW(AssignedClientIDNotification(pMsg));
case M_NTFY_CALLCONNECTED:
        RETURNNEW(CallConnectedNotification(pMsg));
case M_NTFY_REQUESTCANCELLED:
        RETURNNEW(RequestCancelledNotification(pMsg));
case M_NTFY_SITDETECTED:
        RETURNNEW(SITDetectedNotification(pMsg));
case M_NTFY_CNGDETECTED:
        RETURNNEW(CNGDetectedNotification(pMsg));
case M_NTFY_CALLSDISCONNECTED:
        RETURNNEW(CallsDisconnectedNotification(pMsg));
case M_NTFY_PLAYING:
        RETURNNEW(PlayingNotification(pMsg));
case M_NTFY_PLAYDONE:
        RETURNNEW(PlayDoneNotification(pMsg));
case M_NTFY_STOPPEDPLAYING:
        RETURNNEW(StoppedPlayingNotification(pMsg));
case M_NTFY_RECORDING:
        RETURNNEW(RecordingNotification(pMsg));
case M_NTFY_STOPPEDRECORDING:
        RETURNNEW(StoppedRecordingNotification(pMsg));
case M_NTFY_TSPFAILURE:
case M_NTFY_MSPFAILURE:
case M_NTFY_SSPFAILURE:
        RETURNNEW(FailureNotification(pMsg));
case M_NTFY_RECORDINGHANDEDOFF:
        RETURNNEW(RecordingHandedOffNotification(pMsg));
case M_NTFY_TTSPLAYING:
        RETURNNEW(TTSPlayingNotification(pMsg));
case M_NTFY_TTSDONE:
        RETURNNEW(TTSDoneNotification(pMsg));
case M_NTFY_COMPOSITEPROMPTPLAYING:
        RETURNNEW(CompositePromptPlayingNotification(pMsg));

case M_NTFY_COMPOSITEPROMPTSEGMENTPLAYING:
        RETURNNEW(CompositePromptSegmentPlayingNotification(pMsg));
```

```
case M_NTFY_COMPOSITEPROMPTSEGMENTPLAYDONE:
        RETURNNEW(CompositePromptSegmentPlayDoneNotification(pMsg));

case M_NTFY_COMPOSITEPROMPTPLAYDONE:
        RETURNNEW(CompositePromptPlayDoneNotification(pMsg));
case M_NTFY_COMPOSITEPROMPTCREATED:
        RETURNNEW(CompositePromptCreatedNotification(pMsg));

case M_NTFY_COMPOSITEPROMPTSEGMENTAPPENDED:
        RETURNNEW(CompositePromptSegmentAppendedNotification(pMsg));

case M_NTFY_COMPOSITEPROMPTENDED:
        RETURNNEW(CompositePromptEndedNotification(pMsg));

case M_NTFY_RESUME:
        break;
case M_NTFY_INCOMINGCALLREVOKED:
        RETURNNEW(IncomingCallRevokedNotification(pMsg));
case M_NTFY_RECOGNITIONDONE:
        RETURNNEW(RecognitionDoneNotification(pMsg));
case M_NTFY_PROPERTYSET:
        RETURNNEW(PropertySetNotification(pMsg));
case M_NTFY_PROPERTYGOTTEN:
        RETURNNEW(PropertyGottenNotification(pMsg));
case M_NTFY_PROPERTYNOTSET:
        RETURNNEW(PropertyNotSetNotification(pMsg));
case M_NTFY_PROPERTYNOTGOTTEN:
        RETURNNEW(PropertyNotGottenNotification(pMsg));
case M_NTFY_BEGINSPEECH:
        RETURNNEW(BeginSpeechNotification(pMsg));

case M_NTFY_ENDSPEECH:
        RETURNNEW(EndSpeechNotification(pMsg));
case M_NTFY_DTMF:
        RETURNNEW(DTMFNotification(pMsg));
case M_NTFY_CALLACCEPTED:
        RETURNNEW(CallAcceptedNotification(pMsg));
case M_NTFY_CALLREJECTED:
        RETURNNEW(CallRejectedNotification(pMsg));
case M_NTFY_RECOGNITIONSTARTED:
        RETURNNEW(RecognitionStartedNotification(pMsg));
case M_NTFY_MTSLINKDOWN:
        break;
case M_NTFY_CDRINFO:
        RETURNNEW(CDRInfoNotification(pMsg));
case M_NTFY_OUTBOUNDCHANNELAVAILABLE:
        break;
case M_NTFY_MTSLINKUP:
        break;
case M_NTFY_VUIBROKERLINKDOWN:
        break;
case M_NTFY_VUIBROKERLINKUP:
        break;
case M_NTFY_MESSAGINGFAILURE:
        break;
        //T2S section
```

```
                case M_NTFY_T2S_SERVER_TTS_INITIALIZE:
                        RETURNNEW(T2SSPMessage(pMsg));
                case M_NTFY_T2S_SERVER_TTS_START:
                        RETURNNEW(T2SSPstartMessage(pMsg));

                case M_NTFY_T2S_SERVER_TTS_STOP:
                        RETURNNEW(T2SSPstopMessage(pMsg));
                case M_NTFY_T2S_SERVER_TTS_AUDIO_DATA_SEG:
                        RETURNNEW(T2SSPMessageReply(pMsg));
                case M_NTFY_CREATEDYNAMICGRAMMARRESULT:
                        RETURNNEW(CreateDynamicGrammarResultNotification(pMsg));
                case M_NTFY_DELETEDYNAMICGRAMMARRESULT:
                        RETURNNEW(DeleteDynamicGrammarResultNotification(pMsg));
                case M_NTFY_ADDPHRASETODYNAMICGRAMMARRESULT:
                        RETURNNEW(AddPhraseToDynamicGrammarResultNotification(pMsg));
                case M_NTFY_REMOVEPHRASEFROMDYNAMICGRAMMARRESULT:
                        RETURNNEW(RemovePhraseFromDynamicGrammarResultNotification(pMsg));
                case M_NTFY_QUERYDYNAMICGRAMMARCONTENTSRESULT:
                        RETURNNEW(QueryDynamicGrammarContentsResultNotification(pMsg));
                case M_NTFY_INSERTDYNAMICGRAMMARRESULT:
                        RETURNNEW(InsertDynamicGrammarResultNotification(pMsg));
        }
        RETURNNEW(VPPMessage(pMsg));
}
```

Docket No. 50277-2000

```
/*
 * VRSAPICLient.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.65 $
 * $Date: 2000/12/07 03:24:28 $
 */
static const char* id="$Id: VRSAPICLient.cpp,v 1.65 2000/12/07 03:24:28 Mitsuru Oshima Exp $";
#include <process.h>
//#define NUANCE7
#ifdef NUANCE7
#include "dispatcher.hpp"
#include "vrecserver.hpp"
#include "misc-notifications.hpp"
#include "recognition-notifications.hpp"
#include "parameter-notifications.hpp"
#include "sample-buffer.hpp"
#include "nuance-database-notifications-AG.hpp"
#include "dynagram-notifications-AG.hpp"
#else
#include "dispatcher.hh"
#include "vrecserver.hh"
#include "misc-notifications.hh"
#include "recognition-notifications.hh"
#include "parameter-notifications.hh"
#include "sample-buffer.hh"
#include "nuance-database-notifications-AG.hh"
#include "dynagram-notifications-AG.hh"
#endif
#include "VRSAPIClient.h"
#include "VPPMsg/VPPDefs.h"
#include "NuanceSChannel.h"
#include "MTSLib/MediaChannel.h"
#include "VPPMsg/RecognitionDoneNtfy.h"
#include "MTSLib/RecordingMgr.h"
#include "VPPMsg/RecognitionResults.h"
#include "VPPMsg/RecognitionResult.h"
#include "VPPMsg/Slot.h"
#include "MTSLib/SessionNode.h"
#include "VPPUtils/Monitor.h"
#include "VPPUtils/Synchronized.h"
#include "VPPUtils/ConfigMgr.h"
#include "VPPMsg/CreateDynamicGrammarResultNtfy.h"
#include "VPPMsg/DeleteDynamicGrammarResultNtfy.h"
#include "VPPMsg/AddPhraseToDynamicGrammarResultNtfy.h"
#include "VPPMsg/RemovePhraseFromDynamicGrammarResultNtfy.h"
#include "VPPMsg/QueryDynamicGrammarContentsResultNtfy.h"
#include "VPPMsg/InsertDynamicGrammarResultNtfy.h"
char *VRSAPIClient::cDatabaseKeys[] = {
        "dbroot",
        "dbname",
```

```
        "dbclass",
        "dbserver",
        "dbauth",
        0
};
unsigned int VRSAPIClient::cNextDBID = 1;
VRSAPIClient::VRSAPIClient(NuanceSChannel *pSpeechChannel)
{
        cState = S_NotInitialized;
        cSpeechChannel = pSpeechChannel;
        cAudioFormat = AudioFormatUndefined;
        cDispatcher = 0;
        cRunThread = 0;
        cVRecServer = 0;
        cID = (unsigned int)(-1);
        cRecognitionStoppedEvt = (unsigned long)::CreateEvent(NULL, true, true, NULL);  // must be manually reset
- signaled initially because the first StartRecognition should not wait for this event!
        cDatabaseOpenedEvt = (unsigned long)::CreateEvent(NULL, TRUE, FALSE, NULL);
        cDispatcherThreadStoppedEvt = (unsigned long)::CreateEvent(NULL, TRUE, FALSE, NULL);
        cDB = 0;
        cDBID = cNextDBID++;
        cRequestID = 1;
}
VRSAPIClient::~VRSAPIClient(void)
{
        CloseDatabaseConnection();
        // tell the dispatcher thread to stop
        cDispatcher->Return();
        // wait until it really stops before deleting the recserver
        ::WaitForSingleObject((HANDLE)cDispatcherThreadStoppedEvt, INFINITE);
        // NEVER delete the cDispatcher object;
        // deleting the recserver will take care of it
        delete cVRecServer;
        delete cDispatcher;
        cVRecServer = 0;
        ::CloseHandle((HANDLE)cRecognitionStoppedEvt);
        ::CloseHandle((HANDLE)cDatabaseOpenedEvt);
        ::CloseHandle((HANDLE)cDispatcherThreadStoppedEvt);
        ::CloseHandle((HANDLE)cRunThread);
        cRunThread = 0;
        // Delete any outstanding request nodes
        //
        std::list<RequestNode *>::iterator iter = cRequests.begin();
        while (iter != cRequests.end())
        {
                delete *iter;
                iter++;
        }
}
bool
VRSAPIClient::Setup(unsigned int id, NuanceConfig *config, AudioFormat *pAudioFormat,
        NuanceStatus& status)
{
        cID = id;
//      NuanceConfigPrint(config);
        cDispatcher = new Dispatcher(config, status);
```

```
        if (status != NUANCE_OK)
                return false;
        cAudioFormat = *pAudioFormat;
        cVRecServer = new VRecServer(config, cDispatcher, *this, status);
        if (status != NUANCE_OK)
                return false;
        // Spin off the thread that will run the dispatcher
        //
        unsigned thrdaddr;
        cRunThread = ::_beginthreadex(NULL, 0, _RunThreadFunc, this, 0, &thrdaddr);
        if (!cRunThread)
                return false;
        return true;
}
bool
VRSAPIClient::Teardown(void)
{
        return true;
}
unsigned __stdcall
VRSAPIClient::_RunThreadFunc(void *ptr)
{
        NUAN_SSPLog::diag("VRSAPI thread is %ld, Client %p.", ::GetCurrentThreadId(), (VRSAPIClient *)ptr);

        VRSAPIClient *pThis = (VRSAPIClient *)ptr;
        ::GetCurrentThread();
        return pThis->RunThreadFunc();
}
unsigned
VRSAPIClient::RunThreadFunc(void)
{
        cDispatcher->Dispatch();
        cSpeechChannel->diag("Dispatcher thread exiting for VRSAPIClient %p", this);
        ::SetEvent((HANDLE)cDispatcherThreadStoppedEvt);
        return 0;
}
void NUANCE_MEMBER_FUNCTION
VRSAPIClient::Notify(EventNotification *&en)
{
  Monitor m; // dummy
  SessionNode* session = cSpeechChannel->GetSession();
  Synchronized sync(session != NULL ? session : &m);
        switch (en->GetType())
        {
                case Notification::StartOfSpeech:
                {
#ifdef ENABLE_PERF_MEASURE
                        cSpeechChannel->diag("Time it took Nuance to process samples %ld.", ::GetTickCount() -
cSpeechChannel->GetSamplesTime());
                        cSpeechChannel->SetBOSTime(::GetTickCount());
#endif
                        if (cState != S_Recognizing && cState != S_WaitingForResult)
                        {
                                HandleError(cState, en->GetType());
                                return;
                        }
```

```
                    StartOfSpeechNotification *sos = (StartOfSpeechNotification *)en;
                    StartOfSpeech(sos->GetUtteranceID(), sos->GetSafeIndex(),
                            sos->GetActualIndex());
        }
        break;
                case Notification::EndOfSpeech:
                {
                    if (cState != S_Recognizing && cState != S_WaitingForResult)
                    {
                        HandleError(cState, en->GetType());
                        return;
                    }
                    EndOfSpeechNotification *eos = (EndOfSpeechNotification *)en;
                    EndOfSpeech(eos->GetUtteranceID(), eos->GetIndex());
                }
                break;
                case Notification::PartialResult:
                {
                    if (cState != S_WaitingForResult)
                    {
                        HandleError(cState, en->GetType());
                        return;
                    }
                }
                break;
                case Notification::RecognitionStopped:
                {
                    cSpeechChannel->diag("Recognition stopped for SpeechChannel %p, VRSAPIClient %p.",
cSpeechChannel, this);
                    if (cState != S_WaitingForResult)
                    {
                        HandleError(cState, en->GetType());
                        return;
                    }
                    RecognitionStoppedNotification *rs = (RecognitionStoppedNotification *)en;
                    switch (rs->GetReason())
                    {
                        case RecognitionStoppedNotification::COMPLETED:
                        {
                            FinalResult(rs->GetUtteranceID(), rs->GetRecResult());
                        }
                        break;
                        case RecognitionStoppedNotification::HANG_UP:
                        case RecognitionStoppedNotification::ABORTED:
                        default:
                        {
//                          FinalResult(rs->GetUtteranceID(), rs->GetRecResult());
//                          HandleError(cState, rs->GetType());
                            ResetState();
                        }
                        break;
                    }
                }
                break;
                case Notification::InterpretationStopped:
                {
```

```
                    cSpeechChannel->diag("Interpretation stopped for SpeechChannel %p, VRSAPIClient
%p.", cSpeechChannel, this);
                    if (cState != S_WaitingForResult)
                    {
                            HandleError(cState, en->GetType());
                            return;
                    }
                    InterpretationStoppedNotification *is = (InterpretationStoppedNotification *)en;
                    switch (is->GetReason())
                    {
                            case RecognitionStoppedNotification::COMPLETED:
                            {
                                    FinalInterpretation(is->GetNLResult());
                            }
                            break;
                            case RecognitionStoppedNotification::HANG_UP:
                            case RecognitionStoppedNotification::ABORTED:
                            default:
                            {
//                                   HandleError(cState, is->GetType());
                                    ResetState();
                            }
                            break;
                    }
            }
            break;
            case Notification::Exception:
            {
                    ExceptionNotification *e = (ExceptionNotification *)en;
                    HandleException(e->GetStatus());
    }
    break;
            case Notification::InitializationCompleted:
            {
                    if (cState != S_NotInitialized)
                    {
                            HandleError(cState, en->GetType());
                            return;
                    }
                    InitializationCompletedNotification *ic      = (InitializationCompletedNotification *)en;
                    InitializationDone(ic->GetStatus());

                    OpenDatabaseConnection();
//                  ::WaitForSingleObject((HANDLE)cDatabaseOpenedEvt, 10000L);
            }
            break;
            case Notification::ParameterGotten:
            {
                    ParameterGottenNotification *pg = (ParameterGottenNotification *)en;
                    if (pg->GetDataType() == ParameterGottenNotification::String)
                    {
                            if (!strcmp(pg->GetParameterName(), "vrs.FilenameRecorded"))
                                    cUtteranceFilename = ((StringParameterGottenNotification *)en)-
>GetValue();
                    }
            }
```

```
                break;
                case Notification::ParameterSet:
                        break;
                case Notification::NuanceDBOpened:
                {
                        NuanceDBOpenedNotification *ntfy = (NuanceDBOpenedNotification *)en;
                        ::SetEvent((HANDLE)cDatabaseOpenedEvt);
                }
                break;
                case Notification::NuanceDBClosed:
                {
                        NuanceDBClosedNotification *ntfy = (NuanceDBClosedNotification *)en;
                }
                break;
                case Notification::NuanceDBRecordCreated:
                {
                        NuanceDBRecordCreatedNotification *ntfy = (NuanceDBRecordCreatedNotification *)en;
                        DynamicGrammarCreated(ntfy->GetStatus(), ntfy->GetID());
                }
                break;
                case Notification::NuanceDBRecordDeleted:
                {
                        NuanceDBRecordDeletedNotification *ntfy = (NuanceDBRecordDeletedNotification *)en;
                        DynamicGrammarDeleted(ntfy->GetStatus(), ntfy->GetID());
                }
                break;
                case Notification::NuanceDBRecordModified:
                {
                        NuanceDBRecordModifiedNotification *ntfy = (NuanceDBRecordModifiedNotification
*)en;
                        DynamicGrammarModified(ntfy->GetStatus(), ntfy->GetID());
                }
                break;
                case Notification::NuanceDBRecordContents:
                {
                        NuanceDBRecordContentsNotification *ntfy = (NuanceDBRecordContentsNotification
*)en;
                        DynamicGrammarContents(ntfy->GetStatus(), ntfy->GetID(), ntfy->GetPhraseString());
                }
                break;
                case Notification::DynamicGrammarInserted:
                {
                        DynamicGrammarInsertedNotification *ntfy = (DynamicGrammarInsertedNotification *)en;
                        DynamicGrammarInserted(ntfy->GetStatus(), ntfy->GetID());
                }
                break;
                // Special handling for any other notification types can be added.
                // For example:
                // case Notification::RecognitionAborted:
                //   ...
                // case Notification::WordGrammarSet:
                //   ...
                // case Notification::ClashPronsSet:
                //   ...
                // case Notification::ConsistencyPronsSet:
                //   ...
```

```
                default:
                        cSpeechChannel->diag("Unhandled Nuance event %d in SpeechChannel %p VRSAPIClient
%p", en->GetType(), cSpeechChannel,this);
                        break;
        }
}
void
VRSAPIClient::ResetState(void)
{
        cState = S_Idle;
        ::SetEvent((HANDLE)cRecognitionStoppedEvt);
        cSpeechChannel->diag("Client %p SpeechChannel %p, Set State to idle", this, cSpeechChannel);
        cSpeechChannel->NotifySettled();
}
void
VRSAPIClient::InitializationDone(NuanceStatus status)
{
        cSpeechChannel->info("InitializationDone event for SpeechChannel %p, VRSAPIClient %p",
cSpeechChannel, this);
        ResetState();
}
void
VRSAPIClient::StartOfSpeech(unsigned id, int safe_index, int actual_index)
{
        cSpeechChannel->info("StartOfSpeech event for SpeechChannel %p VRSAPIClient %p. safe_index = %d,
actual_index = %d",
                cSpeechChannel, this, safe_index, actual_index);
        cSpeechChannel->NotifyStartOfSpeech();
}
void
VRSAPIClient::EndOfSpeech(unsigned id, int index)
{
        cSpeechChannel->info("EndOfSpeech event for SpeechChannel %p VRSAPIClient %p, index = %d",
cSpeechChannel, this, index);
        cRecognitionStartTime = ::GetTickCount();
        cSpeechChannel->NotifyEndOfSpeech();
}
bool
VRSAPIClient::FillSlot(Slot *pSlot, NLValue *pNLValue, gm::string& grammar_key)
{
        NLValueType valtype;
        NuanceStatus status;
        if (NLGetValueType(pNLValue, &valtype) != NUANCE_OK) {
                cSpeechChannel->warning("could not get slot value type ");
                return false;
    }
        if (valtype == NL_STRING_VALUE)
        {
                char value[256];
                NLGetStringFromValue(pNLValue, value, 256);
                pSlot->SetStringValue(value);
                if (pSlot->GetName() == "namespace") {
                        grammar_key = value;
                }
                cSpeechChannel->diag("<%s \"%s\"> ", pSlot->GetName().c_str(), value);
```

```
                return true;
        }
        else if (valtype == NL_INT_VALUE)
        {
                int val;
                NLGetIntFromValue(pNLValue, &val);
                pSlot->SetIntValue(val);
                cSpeechChannel->diag("<%s %d> ", pSlot->GetName().c_str(), val);
                return true;
        }
        else if (valtype == NL_STRUCTURE_VALUE)
        {
                // pNLValue contains a structure
                //
                cSpeechChannel->diag("<%s ", pSlot->GetName().c_str());
                int i = 0;
                char name[256];
                while (NLGetIthFeatureNameAndType(pNLValue, i++, name, 256, &valtype) == NUANCE_OK)
                {
                        NLValue *pNLFeatureValue = NLNewValue(&status);
                        if (status == NUANCE_OK &&
                                ((status = NLGetFeatureValue(pNLValue, name, pNLFeatureValue)) ==
NUANCE_OK))
                        {
                                Slot *pSubSlot = new Slot(name, 100);
                                if (FillSlot(pSubSlot, pNLFeatureValue, grammar_key))
                                        pSlot->AddStructSlotValue(pSubSlot);
                                else
                                        delete pSubSlot;
                                NLFreeValue(pNLFeatureValue);
                        }
                }
                cSpeechChannel->diag(">");
                return true;
        }
        return false;
}
RecognitionResult *
VRSAPIClient::ComposeRecognitionResult(RecResult const *rrp, int answer,
        NLResult const *pNLResult)
{
        RecognitionResult *pResult = new RecognitionResult;
        char buf[257];
        if (RecResultString(rrp, answer, buf, 256) == NUANCE_OK)
                pResult->SetResult(buf);
        int confidence;
        if (rrp)
                RecResultOverallConfidenceWithoutFiller(rrp, answer, &confidence);
        else
                confidence = 100;
        pResult->SetConfidence( float(confidence) / 100.0);
        NuanceStatus status;
        if (rrp)
        {
                pNLResult = NLInitializeResult(&status);
                if (status != NUANCE_OK)
```

```
        int answers;
        NuanceStatus status = RecResultNumAnswers(rrp, &answers);
        for (int answer = 0; status == NUANCE_OK && answer < answers; answer++)
        {
                cSpeechChannel->info("Result %d: ", answer);
                RecognitionResult *pResult = ComposeRecognitionResult(rrp, answer, 0);
                if (pResult)
                        results.AddResult(pResult);
                cSpeechChannel->info("");
        }
        cSpeechChannel->NotifyDoneRecognizing(&results,
                RecognitionDoneNotification::RS_SuccessfulSpeech);
  SessionNode* session = cSpeechChannel->GetSession();
        // no need to check for session->IsGoingAway()
        if (session)
                session->cSessionLogger.LogItem("RECOGNITION_DELAY", (float)(elapsed / 1000.0));
}
void
VRSAPIClient::FinalInterpretation(NLResult const *pNLResult)
{
        cSpeechChannel->info("FinalInterpretation event for "
                "SpeechChannel %p VRSAPIClient %p", cSpeechChannel, this);
        ResetState();
        // Create a RecognitionResults object from Nuance's NLResults
        // stuff
        //
        RecognitionResults results;
        RecognitionResult *pResult = ComposeRecognitionResult(0, 0, pNLResult);
        if (pResult)
                results.AddResult(pResult);
        cSpeechChannel->info("");

        cSpeechChannel->NotifyDoneRecognizing(&results,
                RecognitionDoneNotification::RS_SuccessfulDTMF);
}
void
VRSAPIClient::HandleException(NuanceStatus status)
{
        cSpeechChannel->warning("Exception event for SpeechChannel %p VRSAPIClient %p, NuanceStatus = %d
(%s)",
                cSpeechChannel, this, status, NuanceStatusMessage(status));
        if (cState == S_Recognizing)
        {
                // Tried to recognize with an invalid grammar.  Notify the client.
                //
                cSpeechChannel->StopRecognition();
                cSpeechChannel->NotifyDoneRecognizing(0, RecognitionDoneNotification::RS_Error);
        }
        ResetState();
}
void
VRSAPIClient::HandleError(RecognitionState state, Notification::Type type)
{
        cSpeechChannel->warning("Error event for SpeechChannel %p VRSAPIClient %P, NotificationType = %d,
state = %d",
                cSpeechChannel, this, type, state);
```

```
        ResetState();
}
void
VRSAPIClient::DynamicGrammarCreated(NuanceStatus status, unsigned int id)
{
        RequestNode *pNode = FindRequestNode(id);
        if (pNode)
        {
                cSpeechChannel->NotifyDynamicGrammarCreated(status, pNode->cDBKey);
                RemoveRequestNode(id);
        }
}
void
VRSAPIClient::DynamicGrammarDeleted(NuanceStatus status, unsigned int id)
{
        RequestNode *pNode = FindRequestNode(id);
        if (pNode)
        {
                cSpeechChannel->NotifyDynamicGrammarDeleted(status, pNode->cDBKey);
                RemoveRequestNode(id);
        }
}
void
VRSAPIClient::DynamicGrammarModified(NuanceStatus status, unsigned int id)
{
        RequestNode *pNode = FindRequestNode(id);
        if (pNode)
        {
                if (pNode->cAddition)
                        cSpeechChannel->NotifyPhraseAddedToDynamicGrammar(status, pNode->cDBKey,
                                pNode->cPhraseID);
                else
                        cSpeechChannel->NotifyPhraseRemovedFromDynamicGrammar(status,
                                pNode->cDBKey, pNode->cPhraseID);
                RemoveRequestNode(id);
        }
}
void
VRSAPIClient::DynamicGrammarContents(NuanceStatus status, unsigned int id,
        const char *phrase_contents)
{
        RequestNode *pNode = FindRequestNode(id);
        if (pNode)
        {
                gm::string str;
                if (phrase_contents)
                        str = phrase_contents;
                else
                        str = gm::string::Nullref();
                cSpeechChannel->NotifyDynamicGrammarContents(status, pNode->cDBKey,
                        pNode->cPhraseID, str);
                RemoveRequestNode(id);
        }
}
void
VRSAPIClient::DynamicGrammarInserted(NuanceStatus status, unsigned int id)
```

```
{
        RequestNode *pNode = FindRequestNode(id);
        if (pNode)
        {
                cSpeechChannel->NotifyDynamicGrammarInserted(status, pNode->cDBKey,
                        pNode->cGrammarName);
                RemoveRequestNode(id);
        }
}
unsigned long
VRSAPIClient:: StartRecognition(const gm::string& grammar)
{
        cSpeechChannel->diag("Client %p SpeechChannel %p, "
                "before waiting for cRecognitionStoppedEvt.",
                this, cSpeechChannel);
        if (WaitForSingleObject((HANDLE)cRecognitionStoppedEvt, 5000) != WAIT_OBJECT_0)
        {
                cSpeechChannel->warning("Client %p SpeechChannel %p, "
                        "StartRecognition failed - previous recognition not stopped yet.",
                        this, cSpeechChannel);
                return SSPF_NOTIDLE;
        }
        cSpeechChannel->diag("Client %p SpeechChannel %p, "
                "after waiting for cRecognitionStoppedEvt.",
                this, cSpeechChannel);
        ::ResetEvent((HANDLE)cRecognitionStoppedEvt);
        if (cState == S_Idle)
        {
                cState = S_Recognizing;
                cSpeechChannel->info("%p->StartUtterance, SpeechChannel %p",
                        this, cSpeechChannel);
                NuanceStatus status = cVRecServer->StartUtterance(grammar, cAudioFormat, cID);
                if (status != NUANCE_OK)
                {
                        cSpeechChannel->warning("%p->StartRecognition failed, "
                                "reason = %s, SpeechChannel %p",
                                this, NuanceStatusMessage(status), cSpeechChannel);
                        ResetState();
                        if (status == NUANCE_UNKNOWN_GRAMMAR)
                                return SSPF_UNKNOWNGRAMMAR;
                        else
                                return SSPF_CANTRECOGNIZE;
                }
                return SSPF_OK;
        }
        else
        {
                cSpeechChannel->error("%p->StartRecognition failed - not idle, "
                        "bug in the code, should have been idle! SpeechChannel %p",
                        this, cSpeechChannel);
                return SSPF_NOTIDLE;
        }
}

unsigned long
VRSAPIClient::PushSamples(const unsigned char *pSamples, unsigned int numbytes)
```

```
{
        if (cState == S_Recognizing)
        {
                cSpeechChannel->info("Client %p SpeechChannel %p - "
                        "Pushing %d samples into recognizer",
                        this, cSpeechChannel, numbytes);
                SampleBuffer sb(cAudioFormat, pSamples, numbytes);
                cSpeechChannel->diag("%p->ProcessSamples SpeechChannel %p",
                        this, cSpeechChannel);
    unsigned long t1 = ::GetTickCount();
                NuanceStatus status = cVRecServer->ProcessSamples(sb);
    cSpeechChannel->diag("Time it took to process rec samples for VRSAPIClient %p SpeechChannel %p: %ld",
this, cSpeechChannel, ::GetTickCount() - t1);
                if (status != NUANCE_OK)
                {
                        cSpeechChannel->warning("ProcessSamples failed for VRSAPIClient %p SpeechChannel
*p, reason = %s", this, cSpeechChannel, NuanceStatusMessage(status));
                        ResetState();
                        return SSPF_CANTRECOGNIZE;
                }
        }
        return SSPF_OK;
}
unsigned long
VRSAPIClient:: StopRecognition(void)
{
        if (cState == S_Recognizing)
        {
                cState = S_WaitingForResult;
                cSpeechChannel->info("%p->EndUtterance, SpeechChannel %p, "
                        "state transitioned from S_Recognizing to S_WaitingForResult",
                        this, cSpeechChannel);
                NuanceStatus status = cVRecServer->EndUtterance();
                cSpeechChannel->diag("%p->EndUtterance returned. "
                        "SpeechChannel %p", this, cSpeechChannel);
                cUtteranceFilename = (const char *)0;
                cVRecServer->GetParameter("vrs.FilenameRecorded", cID);
                return SSPF_OK;
        }
        else
        {
                cSpeechChannel->warning("Client %p SpeechChannel %p "
                        "can't stop recognition because state is not S_Recognizing",
                        this, cSpeechChannel);
                return SSPF_CANTRECOGNIZE;
        }
}
unsigned long
VRSAPIClient::AbortRecognition(void)
{
        if (cState == S_Recognizing)
        {
                cState = S_WaitingForResult;
                cSpeechChannel->info("%p->AbortUtterance, SpeechChannel %p, "
                        "state transitioned from S_Recognizing to S_WaitingForResult",
                        this, cSpeechChannel);
```

```
                return 0;
            status = RecResultNLResult(rrp, answer, (NLResult *)pNLResult);
            if (status != NUANCE_OK)
                return 0;
        }

        cSpeechChannel->diag("Conf=%d, ", confidence);
        int interpretations = NLGetNumberOfInterpretations(pNLResult, &status);
        for (int interp = 0; status == NUANCE_OK && interp < interpretations; interp++)
        {
            NLMakeIthInterpretationActive((NLResult *)pNLResult, interp);
            NLInterpretation *pNLInterpretation = new NLInterpretation();
            int numslots = NLGetNumberOfFilledSlots(pNLResult, &status);
            for (int slot = 0; status == NUANCE_OK && slot < numslots; slot++)
            {
                char name[256];
                NLValueType valtype;
                NLValue *pNLValue = NLNewValue(&status);
                if (status != NUANCE_OK)
                    return 0;
                if (NLGetIthSlotNameAndType(pNLResult, slot, name, 256, &valtype) != NUANCE_OK)
                    return 0;
                status = NLGetSlotValue(pNLResult, name, pNLValue);
                if (status != NUANCE_OK)
                    return 0;
                int nec;
                if (rrp)
                    RecResultConfidenceForIthSlot((RecResult *)rrp, answer, (NLResult
*)pNLResult, slot,
                            &confidence, &nec);
                Slot *pSlot = new Slot(name, (float)(float(confidence)/100.0));

                if (FillSlot(pSlot, pNLValue, pNLInterpretation->GetGrammarKey()))
                    pNLInterpretation->AddSlot(pSlot);
                else
                    delete pSlot;
                NLFreeValue(pNLValue);
            }
            pResult->AddNLInterpretation(pNLInterpretation);
        }
        if (rrp)
            NLFreeResult((NLResult *)pNLResult);
        return pResult;
}
void
VRSAPIClient::FinalResult(unsigned id, RecResult const *rrp)
{
        unsigned long elapsed = ::GetTickCount() - cRecognitionStartTime;
        cSpeechChannel->info("FinalResult event for SpeechChannel %p VRSAPIClient %p, recognition delay =
%.3f", cSpeechChannel, this, (float)(elapsed / 1000.0));
        RecResultPrintToStderr((RecResult *)rrp);
        ResetState();
        // Create a RecognitionResults object from Nuance's RecResults/NLResults
        // stuff
        //
        RecognitionResults results;
```

```
		//unsigned long ret = cVRecServer->AbortUtterance() == NUANCE_OK ? SSPF_OK :
SSPF_CANTRECOGNIZE;
		AbortUttTimer *timer = new AbortUttTimer(*cDispatcher, 0, *this);
		//cSpeechChannel->diag("%p->AbortUtterance returned. "
		//	"SpeechChannel %p", this, cSpeechChannel);
		//return ret;
		return SSPF_OK;
	}
	else
	{
		cSpeechChannel->warning("%p->AbortRecognition SpeechChannel %p "
			"- state is not S_Recognizing", this, cSpeechChannel);
		return SSPF_CANTRECOGNIZE;
	}
}
unsigned long
VRSAPIClient::RecognizeString(const gm::string& grammar, const gm::string& str)
{
	if (cState == S_Idle)
	{
		cState = S_WaitingForResult;
		cSpeechChannel->info("%p->Interpret, SpeechChannel %p",
			this, cSpeechChannel);
		NuanceStatus status = cVRecServer->Interpret(str, grammar, cID);
		cSpeechChannel->diag("%p->Interpret returned, "
			"SpeechChannel %p",this, cSpeechChannel);
		if (status != NUANCE_OK)
		{
			cSpeechChannel->warning("Client %p, SpeechChannel %p, "
				"Interpret failed, reason = %s",
				this, cSpeechChannel, NuanceStatusMessage(status));
			ResetState();
			return SSPF_CANTRECOGNIZE;
		}
		return SSPF_OK;
	}
	else
		return SSPF_NOTIDLE;
}
void
VRSAPIClient::NotifyBoundToSession(SessionNode *pSessionNode)
{
	if (pSessionNode)
	{
		cVRecServer->NewAudioChannel();  // WARNING!!!! this may introduce a significant delay
	}
	else
	{
		if (cState == S_Idle)
			cSpeechChannel->NotifySettled();
	}
}
bool
VRSAPIClient::StartLogging(void)
{
	cVRecServer->SetParameter("vrs.WriteWaveforms", 1, cID);
```

```
        long value = 0;
        gm::string str;
    SessionNode* session = cSpeechChannel->GetSession();
        // no need to check for session->IsGoingAway()
        if (session)
        {
                session->cProperties.GetProperty("Logging.WriteEndpointed", value);
                cVRecServer->SetParameter("vrs.WriteEndpointed", value, cID);
                session->cProperties.GetProperty("Logging.LogfilePath", str);
                cVRecServer->SetParameter("vrs.RecordDirectory", str.c_str(), cID);
                cVRecServer->SetParameter("ep.WriteWaveformsPath", str.c_str(), cID);
                cVRecServer->SetParameter("vrs.RecordFilename", "utt%02d.wav", cID);
                cVRecServer->SetParameter("vrs.RecordCounter", 1, cID);
        }
        return true;
}
bool
VRSAPIClient::LogUtteranceFilename(void)
{
    SessionNode* session = cSpeechChannel->GetSession();
        // no need to check for session->IsGoingAway()
        if (session)
                return session->cSessionLogger.LogItem("UTTERANCE_FILENAME", cUtteranceFilename);
        return true;
}
bool
VRSAPIClient::StopLogging(void)
{
        cVRecServer->SetParameter("vrs.WriteWaveforms", 0, cID);
        return true;
}
void
VRSAPIClient::NotifyPropertySet(Property& prop)
{
        if (prop.GetName() == "Speech.EndpointerBeginSpeechTimeout")
        {
                cSpeechChannel->info("Setting ep.StartSeconds to %ld ms", prop.GetLongValue());
                cVRecServer->SetParameter("ep.StartSeconds", (float)prop.GetLongValue() / 1000L, cID);
        }
        else if (prop.GetName() == "Speech.EndpointerEndSpeechTimeout")
        {
                cSpeechChannel->info("Setting ep.EndSeconds to %ld ms", prop.GetLongValue());
                cVRecServer->SetParameter("ep.EndSeconds", (float)prop.GetLongValue() / 1000L, cID);
        }
}
void
VRSAPIClient::GetUtteranceFilename(gm::string& filename)
{
        filename = cUtteranceFilename;
}
void
VRSAPIClient::AttachRecordingFilename(const gm::string& filename)
{
        cUtteranceFilename += ", " + filename;
}
bool
```

```
VRSAPIClient::OpenDatabaseConnection(void)
{
	if (!cDB)
	{
		gm::string str;
		if (!ConfigMgr::GetString(MTS_REGISTRY_BASE ".Speech Service Providers.Nuance.Database",
			"Provider", str))
			return false;
		cDB = NewDBDescriptor(str);
		if (cDB)
		{
			int i = 0;
			while (cDatabaseKeys[i])
			{
				if (ConfigMgr::GetString(MTS_REGISTRY_BASE ".Speech Service
Providers.Nuance.Database",
					cDatabaseKeys[i], str))
					DBDescriptorAddKV(cDB, cDatabaseKeys[i], str);
				i++;
			}
			return (cVRecServer->OpenDynamicGrammarDatabase(cDB, cDBID) == NUANCE_OK);
		}
	}
	return false;
}
bool
VRSAPIClient::CloseDatabaseConnection(void)
{
	if (cDB)
	{
		cVRecServer->CloseDynamicGrammarDatabase(cDBID);
		::DeleteDBDescriptor(cDB);
		cDB = 0;
		return true;
	}
	return false;
}
unsigned long
VRSAPIClient::CreateDynamicGrammar(const gm::string& db_key, const gm::string& gsl)
{
	NuanceStatus stat;
    cSpeechChannel->diag("CreateDynamicGrammar: key=%s, gsl=%s",
		db_key.c_str(), gsl.c_str());
	if (!cDB)
		return SSPF_DATABASEERROR;
	unsigned int id = AddRequestNode(db_key);
	if (gsl)
		stat = cVRecServer->NewDynamicGrammar(cDBID, db_key, id, gsl, NULL, 1);
	else
		stat = cVRecServer->NewDynamicGrammar(cDBID, db_key, id, 1);
	if (stat != NUANCE_OK)
	{
		RemoveRequestNode(id);
		return SSPF_DATABASEERROR;
	}
	return SSPF_OK;
```

```
}
unsigned long
VRSAPIClient::DeleteDynamicGrammar(const gm::string& db_key)
{
	NuanceStatus stat;
	cSpeechChannel->diag("DeleteDynamicGrammar: key=%s", db_key.c_str());
	if (!cDB)
		return SSPF_DATABASEERROR;
	unsigned int id = AddRequestNode(db_key);
	if ((stat = cVRecServer->DeleteDynamicGrammar(cDBID, db_key, 0)) != NUANCE_OK)
	{
		RemoveRequestNode(id);
		return SSPF_DATABASEERROR;
	}
	return SSPF_OK;
}
unsigned long
VRSAPIClient::AddPhraseToDynamicGrammar(const gm::string& db_key,
	const gm::string& phrase_id, const gm::string& phrase_text,
	const gm::string& phrase_nl, const unsigned long probability)
{
	NuanceStatus stat;
	cSpeechChannel->diag("AddPhraseToDynamicGrammar: key=%s",
		db_key.c_str());
	if (!cDB)
		return SSPF_DATABASEERROR;
	unsigned int id = AddRequestNode(db_key, phrase_id, true);
	if ((stat = cVRecServer->AddPhraseToDynamicGrammar(cDBID, db_key, id, phrase_id, phrase_text,
			phrase_nl, probability / 100.0, 0)) != NUANCE_OK)
	{
		RemoveRequestNode(id);
		return SSPF_DATABASEERROR;
	}
	return SSPF_OK;
}
unsigned long
VRSAPIClient::RemovePhraseFromDynamicGrammar(const gm::string& db_key,
	const gm::string& phrase_id)
{
	NuanceStatus stat;
	if (!cDB)
		return SSPF_DATABASEERROR;
	cSpeechChannel->diag("RemovePhraseFromDynamicGrammar: key=%s",
		db_key.c_str());
	unsigned int id = AddRequestNode(db_key, phrase_id, false);
	if ((stat = cVRecServer->RemovePhraseFromDynamicGrammar(cDBID, db_key, id, phrase_id, 0)) !=
NUANCE_OK)
	{
		RemoveRequestNode(id);
		return SSPF_DATABASEERROR;
	}
	return SSPF_OK;
}
unsigned long
VRSAPIClient::QueryDynamicGrammarContents(const gm::string& db_key,
	const gm::string& phrase_id)
```

```
{
	NuanceStatus stat;
	cSpeechChannel->diag("QueryDynamicGrammarContents: key=%s",
		db_key.c_str());
	if (!cDB)
		return SSPF_DATABASEERROR;
	unsigned int id = AddRequestNode(db_key, phrase_id);
	if (phrase_id)
		stat = cVRecServer->QueryDynamicGrammarContents(cDBID, db_key, id, phrase_id);
	else
		stat = cVRecServer->QueryDynamicGrammarContents(cDBID, db_key, id);
	if (stat != NUANCE_OK)
	{
		RemoveRequestNode(id);
		return SSPF_DATABASEERROR;
	}
	return SSPF_OK;
}
unsigned long
VRSAPIClient::InsertDynamicGrammar(const gm::string& db_key,
	const gm::string& grammar_name)
{
	NuanceStatus stat;
	cSpeechChannel->diag("InsertDynamicGrammar: %s to %s", db_key.c_str(),
		grammar_name.c_str());
	if (!cDB)
		return SSPF_DATABASEERROR;
	unsigned int id = AddRequestNode(db_key, gm::string::Nullref(), false, grammar_name);
	if ((stat = cVRecServer->InsertDynamicGrammar(cDBID, db_key, id, grammar_name, DURATION_CALL))
!= NUANCE_OK)
	{
		cSpeechChannel->warning("Error: InsertDynamicGrammar failed, "
			"VRSAPIClient %p, SpeechChannel %p, status = %d",
			this, cSpeechChannel, stat);
		RemoveRequestNode(id);
		return SSPF_DATABASEERROR;
	}
	return SSPF_OK;
}
unsigned int
VRSAPIClient::AddRequestNode(const gm::string& db_key, const gm::string& phrase_id,
	bool addition, const gm::string& grammar_name)
{
	RequestNode *pNode = new RequestNode;
	pNode->cID = cRequestID++;
	pNode->cDBKey = db_key.c_str();
	pNode->cPhraseID = phrase_id.c_str();
	pNode->cAddition = addition;
	pNode->cGrammarName = grammar_name.c_str();
	cRequests.push_back(pNode);
	return pNode->cID;
}
bool
VRSAPIClient::RemoveRequestNode(unsigned int id)
{
	RequestNode *pNode = FindRequestNode(id);
```

```
        if (pNode)
        {
                cRequests.remove(pNode);
                return true;
        }
        return false;
}
VRSAPIClient::RequestNode *
VRSAPIClient::FindRequestNode(unsigned int id)
{
        std::list<RequestNode *>::iterator iter = cRequests.begin();
        while (iter != cRequests.end())
        {
                if ((*iter)->cID == id)
                        return *iter;
                iter++;
        }
        return 0;
}
#if 0
DynamicGrammarResultNotification::Result
VRSAPIClient::MapNuanceStatus(NuanceStatus stat)
{
        switch(stat)
        {
                case NUANCE_OK:
                        return DynamicGrammarResultNotification::R_Success;
                case NUANCE_UNKNOWN_DATABASE_PROVIDER:
                case NUANCE_DATABASE_PROVIDER_NOT_SUPPORTED:
                case NUANCE_CANNOT_OPEN_DATABASE:
                case NUANCE_DATABASE_PROVIDER_NOT_INSTALLED:
                case NUANCE_DB_HANDLE_ALREADY_IN_USE:
                        return DynamicGrammarResultNotification::R_CantOpenDatabase;
                case NUANCE_NON_EXISTENT_RECORD:
                        return DynamicGrammarResultNotification::R_CantFindDynamicGrammar;
                default:
                        return DynamicGrammarResultNotification::R_Failure;
        }
}
#endif
AbortUttTimer::AbortUttTimer(DispatcherInterface& dispatcher, float interval_ms, VRSAPIClient& vrsapi_client)
: cDisp(dispatcher), cVRSAPIClient(vrsapi_client)
{
 TimeoutTrigger tt(interval_ms);
 cDisp.AddEntry(cTimeTriggerDispEntry, tt, *this, "Timer trigger");
}
void AbortUttTimer::Respond(Trigger const& t, DispatchEntryHandle dispEntry)
{
        if (cVRSAPIClient.cVRecServer && cVRSAPIClient.cVRecServer->AbortUtterance() == NUANCE_OK)
                if (cVRSAPIClient.cSpeechChannel)
                        cVRSAPIClient.cSpeechChannel->debug("AbortUtterance successful.");
        else
                if (cVRSAPIClient.cSpeechChannel)
                        cVRSAPIClient.cSpeechChannel->error("AbortUtterance failed.");
 // self-destruct
 delete this;
```

}

Document: 2020-0026-D

```
/*
 * VUISessionImpl.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.33 $
 * $Date: 2000/12/07 03:19:56 $
 */
static const char* id="$Id: VUISessionImpl.cpp,v 1.33 2000/12/07 03:19:56 Mitsuru Oshima Exp $";
#include <VPPMsg/DropCallMsg.h>
#include <VPPMsg/AcceptCallMsg.h>
#include <VPPMsg/RejectCallMsg.h>
#include <VPPMsg/CancelRequestMsg.h>
#include <VPPMsg/ConnectCallsMsg.h>
#include <VPPMsg/DisconnectCallsMsg.h>
#include <VPPMsg/RecordMsg.h>
#include <VPPMsg/StopRecordMsg.h>
#include <VPPMsg/StartRecognitionMsg.h>
#if 0
#include <VPPMsg/QueryDynamicGrammarContentsMsg.h>
#include <VPPMsg/InsertDynamicGrammarMsg.h>
#include <VPPMsg/CreateDynamicGrammarMsg.h>
#endif
#include <VPPMsg/CreateCompositePromptMsg.h>
#include <VPPMsg/AppendCompositePromptPromptMsg.h>
#include <VPPMsg/AppendCompositePromptSilenceMsg.h>
#include <VPPMsg/AppendCompositePromptDTMFMsg.h>
#include <VPPMsg/AppendCompositePromptTTSMsg.h>
#include <VPPMsg/PlayCompositePromptMsg.h>
#include <VPPMsg/RecognitionResults.h>
#include <VppMsg/RecordingDoneNtfy.h>
#include "SessionMgr.h"
#include "MTSServerImpl.h"
#include "VUISessionImpl.h"
#include "CallTokenMgr.h"
#include "MediaChannel.h"
#include "SpeechChannel.h"
#include "TelephonyChannel.h"
#include "OutboundCallRequest.h"
#include "TelephonyServiceProvider.h"
#include "DynamicGrammar.h"
#include "GrammarHandle.h"
VUISessionImpl::VUISessionImpl(SessionMgr *pSessionMgr, TelephonyChannel *pTChannel) :
  SessionNode(pSessionMgr, pTChannel),
  cPromptState(PROMPT_NONE), cRecognitionState(RECOGNITION_NONE),
  cSpeechStatus(SSPF_OK), cMediaStatus(MSPF_OK), cTelephonyStatus(TSPF_OK),
  cOutdialState(ODS_NONE)
{
}
VUISessionImpl::~VUISessionImpl(void)
{
}
```

```
void
VUISessionImpl::Disconnected() {
        if (IsGoingAway() == false) {
                Synchronized sync(this);
                if (IsGoingAway() == false) {
                        terminate();
                }
                return;
        }
}
void
VUISessionImpl::SetVUIClient(invui::VUIClient_var vc) {
  Synchronized sync(this);
  cVUIClient = vc;
  sync.notify();
}
CallToken&
VUISessionImpl::GetCurrentCallToken() {
  return cTelephonyChannel->GetCurrentCallToken();
}
/*************************************************/
//
// Media Notifications
//
/*************************************************/
CMSMessage::SerialNumberT
VUISessionImpl::NotifyCompositePromptPlayDone(CallToken& callToken)
{
  Synchronized sync(this); // for future use
  if (cPromptState == PROMPT_PLAYING) {
    cPromptState = PROMPT_DONE;
    sync.notifyAll();
  }

        diag("NotifyCompositePromptPlayDone: state %u",(UINT)cPromptState);
  // CompositePromptPlayDoneNotification ntfy(callToken);
  //return WriteWrappedMessage(ntfy);
  return 0;
}
CMSMessage::SerialNumberT
VUISessionImpl::NotifyRecording(CallToken& callToken, RECORDINGID recordingID, const gm::string& filename)
{
  Synchronized sync(this); // for future use
  cRecordResult.filename = CORBA::string_dup(filename);
  // RecordingNotification ntfy(callToken, recordingID, filename);
  //return WriteWrappedMessage(ntfy);
  return 0;
}
//virtual
CMSMessage::SerialNumberT
VUISessionImpl::NotifyRecordingDone(CallToken& callToken, RecordingDoneNotification::RecordingDoneReason
reason, unsigned long duration)
{
  Synchronized sync(this);
  cRecordResult.duration = duration;
  // set the recording done reason
```

```
    switch (reason)
    {
      case RecordingDoneNotification::RDR_NoSpeech:
        cRecordResult.reason = invui::RDR_NoSpeech;
        break;
      case RecordingDoneNotification::RDR_EndOfSpeech:
        cRecordResult.reason = invui::RDR_EndOfSpeech;
        break;
      case RecordingDoneNotification::RDR_TooMuchSpeech:
        cRecordResult.reason = invui::RDR_TooMuchSpeech;
        break;
      case RecordingDoneNotification::RDR_Error:
        cRecordResult.reason = invui::RDR_Error;
        break;
      default:
        cRecordResult.reason = invui::RDR_Error;
        break;
    }

    cRecordingState = RECORDING_DONE;
    sync.notifyAll();
    return 0;
  }
  CMSMessage::SerialNumberT
  VUISessionImpl::NotifyCompositePromptSegmentPlayDone(CallToken& callToken)
  {
    Synchronized sync(this); // for future use
    cPlayAndRecResults.play_result.number_of_played_prompt++;
    // CompositePromptSegmentPlayDoneNotification ntfy(callToken);
    // return WriteWrappedMessage(ntfy);
    return 0;
  }
  CMSMessage::SerialNumberT
  VUISessionImpl::NotifyDTMF(CallToken& callToken, char dtmf, unsigned long timestamp)
  {
    Synchronized sync(this); // for future use
    if (cVUIClient != NULL) {
      invui::Call_var call = new invui::Call();
      call->id = CallTokenMgr::GetCallID(GetCurrentCallToken());
      try {
        call->session = duplicateReference();
        info("NotifyDTMF : %c", dtmf);
        cVUIClient->dtmfPressed(call, dtmf);
      } catch (const CORBA::Exception& e) {
        error("NotifyDTMF got CORBA exception");
        cerr << "VUISessionImpl::NotifyDTMF(1) " << e << endl;
      }
    }
    return 0;
  }
  /************************************************/
  //
  // Telephony Notifications
  //
  /************************************************/
  CMSMessage::SerialNumberT
```

```
VUISessionImpl::NotifyCallProgress(RequestToken& requestToken, CallProgressNotification::CallProgressT
notification)
{
  Synchronized sync(this); // for future use
  OutboundCallRequest* request =
    OutboundCallRequest::GetOutboundCallRequest(requestToken);
  if (cOutdialState == ODS_ALERTING_NOTIFY) {
    diag("call progress in alerting notify state. waiting ..");
    sync.wait(1000);
  }
  if (cOutdialState == ODS_ALERTING &&
    notification != CallProgressNotification::CPN_Alerting) {
    // all events comes after alerting must be error notification.
    // needs to notify client
    info("got error while alerting. cancelling call");
    if (cVUIClient == NULL) {
      sync.wait(5000); // just 5 sec.
    }
    if (cVUIClient != NULL) {
      invui::Call_var call = new invui::Call();
      call->id = CallTokenMgr::GetCallID(GetCurrentCallToken());
      try {
        call->session = duplicateReference();
        info("NotifyCallProgress ");
        cVUIClient->callCancelled(call);
      } catch (const CORBA::Exception& e) {
        error("NotifyCallProgress : got CORBA exception");
        cerr << "VUISessionImpl::NotifyCallProgress(1) " << e << endl;
      }
    }
    cSessionMgr->DestroySession(cTelephonyChannel);
    return 0;
  }
  if (notification != CallProgressNotification::CPN_Alerting) {
    warning("call failed.");
    request->CallFailure(notification);
    cSessionMgr->DestroySession(cTelephonyChannel);
    return 0;
  } else if (request == NULL) {
    // client timed out and will be cancelled it.
    return 0;
  }
  try {
    CallToken& callToken = GetCurrentCallToken();
    cID = callToken.GetSequenceNumber();
    setLogID("[%ld]",cID);
    UpdateCallToken();
    // this is return value. dont' free it.
    invui::Call_ptr call = new invui::Call();
    call->session = duplicateReference();
    call->id = callToken.GetSequenceNumber();
    CallTokenMgr::Register(call->id, callToken);
    cOutdialState = ODS_ALERTING_NOTIFY;
    diag("call alerting");
    request->CallAlerting(call);
  } catch (const CORBA::Exception& e) {
```

```
    cerr << "VUISessionImpl::NotifyCallProgress(2) " << e << endl;
  }
  return 0;
/*
  // when error happends
  CallProgressNotification ntfy(request, notification);
//  return WriteWrappedMessage(ntfy);
  return 0;
*/
}
CMSMessage::SerialNumberT
VUISessionImpl::NotifyCallConnected(RequestToken& requestToken, CallToken& callToken)
{
  Synchronized sync(this);
  // UpdateCallToken();
  // CDR_LOG - this is a billable event (inbound call accepted).
  int count = 5;
  while (cVUIClient == NULL && count-- >0) {
    sync.wait(500);
  }
  if (cVUIClient == NULL) {
    if (IsGoingAway() == false) {
      terminate();
    }
  } else {
    try {
      if (cOutdialState == ODS_NONE) {
        // connected without Altering
        diag("connected without Alerting");
        OutboundCallRequest* request =
          OutboundCallRequest::GetOutboundCallRequest(requestToken);
        CallToken& callToken = GetCurrentCallToken();
        cID = callToken.GetSequenceNumber();
        setLogID("[%ld]",cID);
        UpdateCallToken();
        // this is return value. dont' free it.
        invui::Call_ptr call = new invui::Call();
        call->session = duplicateReference();
        call->id = callToken.GetSequenceNumber();
        CallTokenMgr::Register(call->id, callToken);
        if (request == NULL) {
          info("no one is waiting for call, cancelled");
          cancelCall(*call);
          delete call; // not return value
          return 0;
        } else {
          diag("alerting ..");
          cOutdialState = ODS_ALERTING_NOTIFY;
          request->CallAlerting(call);
          sync.wait(300); // for now.
        }
      }
      invui::Call_var call = new invui::Call();
      call->id = CallTokenMgr::GetCallID(callToken);
      call->session = duplicateReference();
      cVUIClient->callConnected(call);
```

```
    } catch (const CORBA::Exception &e) {
      cerr << "VUISessionImpl::NotifyCallConnected " << e << endl;
      // terminating
      terminate();
    }
  }
  return 0;
}
CMSMessage::SerialNumberT
VUISessionImpl::NotifyCallDropped(CallToken& ct)
{
  Synchronized sync(this);
  CMSMessage::SerialNumberT ret = 0;
  CORBA::LongLong id = CallTokenMgr::GetCallID(ct);
  if (id != 0 && cVUIClient != NULL) {
    invui::Call call;
    call.id = id;
    try {
      call.session = duplicateReference();
      cVUIClient->callDropped(call);
    } catch (CORBA::Exception &ex) {
      cerr << "VUISessionImpl::NotifyCallDropped " << ex << endl;
      // notifying VUIClient which does not exist.
    }
  }
  // I don't know what these are doing... M.O
//  ret = WriteWrappedMessage(ntfy);
  // no need to check for session->IsGoingAway()
  // CDR_LOG - this is a billable event (inbound call accepted).
  return ret;
}
void
VUISessionImpl::NotifyIncomingCallRevoked(CallToken& ct)
{
  Synchronized sync(this);
  CORBA::LongLong id = CallTokenMgr::GetCallID(ct);
  if (id != 0 && cVUIClient != NULL) {
    invui::Call call;
    call.id = id;
    try {
      call.session = duplicateReference();
      cVUIClient->callDropped(call);
    } catch (CORBA::Exception &ex) {
      cerr << "VUISessionImpl::NotifyIncomingCallRevoked " << ex << endl;
    }
  }
/*
  IncomingCallRevokedNotification ntfy(ct);
  MTSServerNode *pServerNode = cServer->FindServerNodeForClientID(VUIClientID);
  if (pServerNode)
    ret = pServerNode->WriteWrappedMessage(ntfy, VUIClientID);
*/
  // do not check for session->IsGoingAway() because it is going away
  cSessionMgr->DestroySession(cTelephonyChannel);
}
/*************************************************/
```

```
//
// Speech Notification
//
/*************************************************/
static char* CopySlots(invui::Slot& slot, Slot* n_slot) {
   switch (n_slot->GetType()) {
   case Slot::T_String: {
      // slot._d(invui::T_String);
      invui::Slot::sslot_s sslot;
                  if (n_slot->GetName() == NULL) {
                        sslot.name = CORBA::strdup("");
                  } else {
                        sslot.name = CORBA::strdup(n_slot->GetName());
                  }
      sslot.value = CORBA::strdup(n_slot->GetString());
      sslot.confidence = n_slot->GetConfidence();
      slot.sslot(sslot);
/*
      if (strcmp(sslot.name, "namespace") == 0) {
         return sslot.value;
//          ivRes.grammar = CORBA::strdup(sslot.value);
      }
 */
      break;
   }
   case Slot::T_Int: {
      //slot._d(invui::T_Int);
      invui::Slot::islot_s islot;
                  if (n_slot->GetName() == NULL) {
                        islot.name = CORBA::strdup("");
                  } else {
               islot.name = CORBA::strdup(n_slot->GetName());
                  }
      islot.value = n_slot->GetInt();
      islot.confidence = n_slot->GetConfidence();
      slot.islot(islot);
      break;
   }
   case Slot::T_Structure: {
      //slot._d(invui::T_Structure);
      invui::Slot::rslot_s rslot;

      int count = 0;
      for (int l=0;
         n_slot->GetStructSlot(l) != NULL;
         l++) {
         count ++;
      }
      rslot.name = CORBA::string_dup("");
      rslot.child_slot.length(count);

      Slot *pSubSlot;
      for (l=0;
         (pSubSlot = n_slot->GetStructSlot(l)) != NULL;
         l++) {
         CopySlots(rslot.child_slot[l], pSubSlot);
```

```
    }
    /*
     rslot.name = n_slot->GetName();
     rslot.value = n_slot->GetInt();
     rslot.confidence = n_slot->GetConfidence();
     */
    slot.rslot(rslot);
    break;
  }
  }
  return NULL;
}
CMSMessage::SerialNumberT
VUISessionImpl::NotifyRecognitionDone(CallToken& ct,
                        RecognitionResults *pResults,
                        RecognitionDoneNotification::RecognitionStatus status)
{
  Synchronized sync(this); // for future use
  if (cRecognitionState != RECOGNITION_STARTED) {
    return 0;
  }
  switch (status) {
  case RecognitionDoneNotification::RS_SuccessfulSpeech:
    cPlayAndRecResults.rec_result.status = invui::RS_SuccessfulSpeech;
    break;
  case RecognitionDoneNotification::RS_SuccessfulDTMF:
    cPlayAndRecResults.rec_result.status = invui::RS_SuccessfulDTMF;
    debug("recognition done : status = SuccessfulDTMF");
    break;
  case RecognitionDoneNotification::RS_NoSpeech:
    cPlayAndRecResults.rec_result.status = invui::RS_NoSpeech;
    debug("recognition done : status = NoSpeech");
    break;
  case RecognitionDoneNotification::RS_TooMuchSpeech:
    cPlayAndRecResults.rec_result.status = invui::RS_TooMuchSpeech;
    debug("recognition done : status = TooMuchSpeech");
    break;
  case RecognitionDoneNotification::RS_CallDropped:
    cPlayAndRecResults.rec_result.status = invui::RS_CallDropped;
    debug("recognition done : status = CallDropped");
    break;
  case RecognitionDoneNotification::RS_Error:
    cPlayAndRecResults.rec_result.status = invui::RS_Error;
    error("recognition done : status = Error");
    break;
  }
  if (status == RecognitionDoneNotification::RS_SuccessfulSpeech ||
    status == RecognitionDoneNotification::RS_SuccessfulDTMF) {
    unsigned short num_results = pResults->GetNumResults();
    cPlayAndRecResults.rec_result.results.length(num_results);
    debug("recognition done : status = SuccessfulSpeech, num_result=%d",
        num_results);
    for (int i=0; i<num_results; i++) {
      RecognitionResult* result = pResults->GetResult(i);
      invui::RecognitionResult ivRes;
      //ivRes.grammar = CORBA::string_dup(result->grammar_key);
```

```
        if (result->GetResult() == gm::string::Nullref()) {
          ivRes.result = CORBA::string_dup("");
        } else {
          ivRes.result = CORBA::string_dup(result->GetResult());
        }
        ivRes.confidence = result->GetConfidence();

        debug("result[%d] : result = %s, confidence=%f",
            i, (const char*)ivRes.result,
            (double)ivRes.confidence);
        int num_nli = result->GetNumNLInterpretations();
        ivRes.nl_interpretations.length(num_nli);

        for (int j=0; j<num_nli; j++) {
          NLInterpretation* nli = result->GetNLInterpretation(j);

          int num_slots = nli->GetNumSlots();

          ivRes.nl_interpretations[j].slots.length(num_slots);
                                    if (nli->GetGrammarKey() == NULL) {
                                            ivRes.nl_interpretations[j].grammar = CORBA::string_dup("");
                                    } else {
                                            ivRes.nl_interpretations[j].grammar = CORBA::string_dup(nli-
>GetGrammarKey());
                                    }
          for (int k=0; k<num_slots; k++) {
            CopySlots(ivRes.nl_interpretations[j].slots[k], nli->GetSlot(k));
          }
        }
        cPlayAndRecResults.rec_result.results[i] = ivRes;
      }
    } else {
      cPlayAndRecResults.rec_result.results.length(0);
    }

    cRecognitionState = RECOGNITION_DONE;
    sync.notifyAll();
    //return WriteWrappedMessage(ntfy);
    return 0;
}
/*************************************************/
//
// Utility Function
//
/*************************************************/
invui::VUISession_ptr
VUISessionImpl::duplicateReference() {
  // Activate it on the default POA which is root POA for this servant
#if 1
  MTSServerImpl* mtsserver = cSessionMgr->GetMTSServerImpl();
  PortableServer::POA_var default_poa = mtsserver->_default_POA();
  CORBA::Object_var ref = default_poa->servant_to_reference(this);
  invui::VUISession_var session = invui::VUISession::_narrow(ref);
  return invui::VUISession::_duplicate(session);
#else
  return _this();
```

```
#endif
}
void
VUISessionImpl::deactivate() {
  MTSServerImpl* mtsserver = cSessionMgr->GetMTSServerImpl();
  PortableServer::POA_var default_poa = mtsserver->_default_POA();
  CORBA::Object_var ref = default_poa->servant_to_reference(this);
  CORBA::String_var ior = mtsserver->GetORB()->object_to_string(ref);
  PortableServer::ObjectId_var id = PortableServer::string_to_ObjectId(ior);
  default_poa->deactivate_object(id);
}
CMSMessage::SerialNumberT
VUISessionImpl::NotifyMSPFailure(CallToken& ct,
                    CMSMessage::SerialNumberT FailureSerialNo,
                    unsigned long Info)
{
  Synchronized sync(this); // for future use
  cMediaStatus = Info;
  sync.notifyAll();
  //FailureNotification ntfy(ct, M_NTFY_MSPFAILURE, FailureSerialNo, Info);
  //return WriteWrappedMessage(ntfy);
  return 0;
}
CMSMessage::SerialNumberT
VUISessionImpl::NotifySSPFailure(CallToken& ct,
                    CMSMessage::SerialNumberT FailureSerialNo,
                    unsigned long Info)
{
  Synchronized sync(this); // for future use
  cPlayAndRecResults.rec_result.status = invui::RS_Error;
  cSpeechStatus = Info;
  cRecognitionState = RECOGNITION_NONE;
  sync.notifyAll();
  // FailureNotification ntfy(ct, M_NTFY_SSPFAILURE, FailureSerialNo, Info);
  //return WriteWrappedMessage(ntfy);
  return 0;
}
CMSMessage::SerialNumberT
VUISessionImpl::NotifyTSPFailure(CallToken& ct,
                    CMSMessage::SerialNumberT FailureSerialNo,
                    unsigned long Info)
{
  Synchronized sync(this); // for future use
  cTelephonyStatus = Info;
  sync.notifyAll();
  // FailureNotification ntfy(ct, M_NTFY_TSPFAILURE, FailureSerialNo, Info);
  //return WriteWrappedMessage(ntfy);
  return 0;
}
CMSMessage::SerialNumberT
VUISessionImpl::NotifyTSPFailure(CMSMessage::SerialNumberT FailureSerialNo,
                    unsigned long Info)
{
  CallToken invalid;
  Synchronized sync(this); // for future use
  cTelephonyStatus = Info;
```

```
    sync.notifyAll();
    //FailureNotification ntfy(invalid, M_NTFY_TSPFAILURE, FailureSerialNo, Info);
    //return WriteWrappedMessage(ntfy);
    return 0;
}
/**
 * Util
 */
static void
GetGrammarHandle(const invui::GrammarHandle& handle, GrammarHandle& to_handle) {
    if (handle._d() == invui::DYNAMIC) {
        to_handle.cHandleType = GrammarHandle.DYNAMIC;
        to_handle.key = (char*)handle.key();
    } else {
        to_handle.cHandleType = GrammarHandle.BUILTIN;
        to_handle.key = (char*)handle.name();
    }
}
/**
 *
 * CORBA METHODS
 *
 */
void
VUISessionImpl::clearPrompts() {
    Synchronized sync(this);
    diag("clearPrompts");
    CheckValid();
    if (cPromptState != PROMPT_NONE) {
        error("prompt stat is not PROMPT_NONE");
        throw invui::MediaError(MSPF_NOTIDLE);
    }
    CreateCompositePromptMessage msg(GetCurrentCallToken());
    cPlayMediaChannel->HandleCreateCompositePrompt(this, &msg, NULL);
}
invui::PlayAndRecResults*
VUISessionImpl::playAndRecognize(const invui::GrammarHandle& handle,
                        invui::TIME bos_timeout,
                        invui::TIME tms_timeout,
                        invui::TIME eos_timeout)
{
    Synchronized sync(this);
    diag("playAndRecognize : %s, bos=%ld, tms=%ld, eos=%ld", handle.key(),
        bos_timeout, tms_timeout, eos_timeout);
    CheckValid();
    if (cPromptState != PROMPT_NONE) {
        error("cPromptState is not PROMPT_NONE : %d", cPromptState);
        throw invui::MediaError(MSPF_NOTIDLE);
    } else if (cRecognitionState != RECOGNITION_NONE) {
        error("cRecognitionState is not RECOGNITION_NONE : %d", cRecognitionState);
        throw invui::SpeechError(SSPF_NOTIDLE);
    }
    cRecognitionState = RECOGNITION_STARTED;
    cPlayAndRecResults.rec_result.results.length(0); // free memory
    cProperties.SetProperty("Speech.BeginSpeechTimeout", (long)bos_timeout);
    cProperties.SetProperty("Speech.TooMuchSpeechTimeout", (long)tms_timeout);
```

```
    Property prop("Speech.EndpointerEndSpeechTimeout", (long)eos_timeout);
    SetProperty(prop);
    invui::TIME timeout = 3000000;
    invui::TIME until = ::GetTickCount() + timeout;
    StartPrompts();
    GrammarHandle new_handle;
    GetGrammarHandle(handle, new_handle);
    cSpeechChannel->HandleStartRecognition(this,
                                new_handle,
                                cPlayMediaChannel,
                                cRecordMediaChannel);
    invui::TIME timeLeft = until - ::GetTickCount();
    while ( CheckValid() &&
          CheckMedia() &&
          CheckSpeech() &&
          cRecognitionState != RECOGNITION_DONE &&
          timeLeft >0) {
       sync.wait(timeLeft);
       timeLeft = until - ::GetTickCount();
    }
    if (timeLeft <= 0 &&
       cRecognitionState != RECOGNITION_DONE) {
       cRecognitionState = RECOGNITION_NONE;
       cPromptState = PROMPT_NONE;
       throw invui::SpeechError(SSPF_CANTRECOGNIZE);
    }
    cRecognitionState = RECOGNITION_NONE;
    cPromptState = PROMPT_NONE;
    invui::PlayAndRecResults* results = new invui::PlayAndRecResults();
    results->play_result = cPlayAndRecResults.play_result;
    results->rec_result = cPlayAndRecResults.rec_result;
    return results;
  /*
    invui::PlayAndRecResults* results = new invui::PlayAndRecResults();
    results->play_result;
    invui::RecognitionResults* rec_result = recognize(set, timeout);
    results->rec_result = *rec_result;
    delete rec_result;
    return results;
  */
  }
  invui::PlayResult
  VUISessionImpl::playPrompts() {
    Synchronized sync(this);
    diag("playPrompts");
    CheckValid();
    if (cPromptState != PROMPT_NONE) {
       error("cPromptState is not PROMPT_NONE : %d", cPromptState);
       throw invui::MediaError(MSPF_NOTIDLE);
    }
    StartPrompts();
    while (CheckValid() &&
          CheckMedia() &&
          cPromptState == PROMPT_PLAYING) { // until DONE
       sync.wait();
    }
```

```
    cPromptState = PROMPT_NONE;
    return cPlayAndRecResults.play_result;
/*
    // count it later
    invui::PlayResult result;
    result.number_of_played_prompt = 5;
    return result;
*/
}
invui::PlayResult
VUISessionImpl::playPrompts2(CORBA::Boolean _async)
{
    Synchronized sync(this);
    CheckValid();
    diag("playPrompts2(%u)",UINT(_async));
    if (cPromptState != PROMPT_NONE) {
        error("cPromptState is not PROMPT_NONE : %d", cPromptState);
        throw invui::MediaError(MSPF_NOTIDLE);
    }
    StartPrompts();
            if(!_async)
            {
     while (CheckValid() &&
          CheckMedia() &&
          cPromptState == PROMPT_PLAYING) { // until DONE
        sync.wait();
     } // while
     cPromptState = PROMPT_NONE;
            } // if
    return cPlayAndRecResults.play_result;
} // playPrompts2
void
VUISessionImpl::StartPrompts() {
    // called within synchronized block
    cPromptState = PROMPT_PLAYING;
    cPlayAndRecResults.play_result.number_of_played_prompt = 0;
    PlayCompositePromptMessage msg(GetCurrentCallToken());
    cPlayMediaChannel->HandlePlayCompositePrompt(this, &msg, NULL);
}
bool
VUISessionImpl::CheckTelephony() {
    if (cTelephonyStatus != TSPF_OK) {
        invui::TelephonyError e(cTelephonyStatus);
        error("CheckTelephony: Error = %d", cTelephonyStatus);
        cTelephonyStatus = TSPF_OK;
        throw e;
    }
    return true;
}
bool
VUISessionImpl::CheckMedia() {
    if (cMediaStatus != MSPF_OK) {
        invui::MediaError e(cMediaStatus);
        error("CheckMedia: Error = %d" , cMediaStatus);
        cMediaStatus = MSPF_OK;
        if (cPromptState = PROMPT_PLAYING) {
```

```
            cPromptState = PROMPT_NONE;
        }
        throw e;
    }
    return true;
}
bool
VUISessionImpl::CheckSpeech() {
    if (cSpeechStatus != SSPF_OK) {
        invui::SpeechError e(cSpeechStatus);
        cSpeechStatus = SSPF_OK;
        error("CheckSpeech: Error = %d", cSpeechStatus);
        if (cRecognitionState == RECOGNITION_STARTED) {
            cRecognitionState = RECOGNITION_NONE;
        }
        throw e;
    }
    return true;
}
bool
VUISessionImpl::CheckValid() {
    if (IsGoingAway()) {
        error("Session is going away");
        throw invui::SessionEnded();
    }
    return true;
}
void
VUISessionImpl::appendPrompts(const invui::Prompts& ps){
    diag("appendPrompts : len=%d", ps.length());
    int len = ps.length();
    for (int i=0; i<len; i++) {
        appendPrompt(ps[i]);
    }
}
void
VUISessionImpl::appendPrompt(const invui::Prompt& p) {
    CallToken ct = GetCurrentCallToken();
    switch (p._d()) {
    case invui::T_TTS: {
        const char* tts = p.tts();
        AppendCompositePromptTTSMessage msg(ct, tts, "", 0, 0, 0);

        cPlayMediaChannel->HandleAppendCompositePromptTTS(this, &msg, NULL);
        break;
    }
    case invui::T_AUDIO: {
        invui::Prompt::audio_s audio = p.audio();

        gm::string alt_tts;
        const char* tts = (const char*)audio.alternative_tts;
        if (strlen(tts)==0) {
            alt_tts = gm::string::Nullref();
        } else {
            alt_tts = tts;
        }
```

```
    AppendCompositePromptPromptMessage msg(ct,
                                (const char*)audio.prompt,
                                alt_tts,
                                true);
    cPlayMediaChannel->HandleAppendCompositePromptPrompt(this, &msg, NULL);
    break;
  }
  case invui::T_AUDIO_ABS: {
    invui::Prompt::audio_abs_s audio = p.audio_abs();
    gm::string alt_tts;
    const char* tts = (const char*)audio.alternative_tts;
    if (strlen(tts)==0) {
      alt_tts = gm::string::Nullref();
    } else {
      alt_tts = tts;
    }

    AppendCompositePromptPromptMessage  msg(ct,
                                (const char*)audio.prompt,
                                alt_tts,
                                false);

    cPlayMediaChannel->HandleAppendCompositePromptPrompt(this, &msg, NULL);
  }
                  break;
  case invui::T_SILENCE: {
    AppendCompositePromptSilenceMessage msg(ct, p.length());
    cPlayMediaChannel->HandleAppendCompositePromptSilence(this, &msg, NULL);
    break;
  }
  case invui::T_DTMF: {
    const char *dtmf = p.dtmf();
    AppendCompositePromptDTMFMessage msg(ct, dtmf);
    cPlayMediaChannel->HandleAppendCompositePromptDTMF(this, &msg, NULL);
    break;
  }
  }
  CheckMedia();
}
invui::GrammarHandle*
VUISessionImpl::getGrammarHandle(const invui::DynamicGrammar& gram) {
  Synchronized sync(this);
  diag("getGrammarHandle: %s",
                  gram._d() == invui::DG_TEXT ? (const char*)gram.text().key : "set");
  CheckValid();
  GrammarHandle handle;
  unsigned long ret;
  switch (gram._d()) {
  case invui::DG_TEXT: {
    gm::string grammar = (const char*)gram.text().grammar;
    gm::string key = (const char*)gram.text().key;
    TextGrammar text(grammar, key);
    ret = cSpeechChannel->GetGrammarHandle(text, handle);
    break;
  }
  case invui::DG_SET: {
```

```
    GrammarSet set;
    invui::GrammarSet iset = gram.set();
    for (int i=0; i<iset.length(); i++) {
      GetGrammarHandle(iset[i].handle, handle);
      set.add(handle, iset[i].weight);
    }
    ret = cSpeechChannel->GetGrammarHandle(set, handle);
    break;
  }
  }
  if (ret != SSPF_OK) {
    throw invui::SpeechError(ret);
  }
  invui::GrammarHandle* invui_handle = new invui::GrammarHandle();
  switch (handle.cHandleType) {
  case GrammarHandle::DYNAMIC:
    invui_handle->key(CORBA::string_dup(handle.key));
    break;
  case GrammarHandle::BUILTIN:
    invui_handle->name(CORBA::string_dup(handle.key));
    break;
  };
  return invui_handle;
}
void
VUISessionImpl::addToOutput(const invui::Call& c) {
}
void
VUISessionImpl::addToInput(const invui::Call& c) {
}
void
VUISessionImpl::removeFromInput(const invui::Call& c) {
}
void
VUISessionImpl::removeFromOutput(const invui::Call& c) {
}
invui::RecognitionResults*
VUISessionImpl::recognize(const invui::GrammarHandle& handle,
                          invui::TIME bos_timeout,
                          invui::TIME tms_timeout,
                          invui::TIME eos_timeout)
{
  Synchronized sync(this);
  diag("recognize : %s, bos=%ld, tms=%ld, eos=%ld", (const char*)handle.key(),
     bos_timeout, tms_timeout, eos_timeout);
  CheckValid();
  CheckMedia();

  if (cRecognitionState != RECOGNITION_NONE) {
    error("cRecognitionState is not RECOGNITION_NONE : %d", cRecognitionState);
    throw invui::SpeechError(SSPF_NOTIDLE);
  }
  cProperties.SetProperty("Speech.BeginSpeechTimeout", (long)bos_timeout);
  cProperties.SetProperty("Speech.TooMuchSpeechTimeout", (long)tms_timeout);
  Property prop("Speech.EndpointerEndSpeechTimeout", (long)eos_timeout);
  SetProperty(prop);
```

```
    invui::TIME timeout = 3000000;
    invui::TIME until = ::GetTickCount() + timeout;
    invui::TIME timeLeft = timeout;
    cRecognitionState = RECOGNITION_STARTED;
    GrammarHandle new_handle;
    GetGrammarHandle(handle, new_handle);
    cSpeechChannel->HandleStartRecognition(this,
                                new_handle,
                                cPlayMediaChannel,
                                cRecordMediaChannel);
    timeLeft = until - ::GetTickCount();
    while ( CheckValid() &&
          CheckMedia() &&
          CheckSpeech() &&
          cRecognitionState != RECOGNITION_DONE &&
          timeLeft >0) {
       sync.wait(timeLeft);
       timeLeft = until - ::GetTickCount();
    }
    if (timeLeft <= 0 &&
       cRecognitionState != RECOGNITION_DONE) {

       cRecognitionState = RECOGNITION_NONE;
       cPromptState = PROMPT_NONE;

       throw invui::SpeechError(SSPF_CANTRECOGNIZE);
    }
    cRecognitionState = RECOGNITION_NONE;
    cPromptState = PROMPT_NONE;
    invui::RecognitionResults* results = new invui::RecognitionResults();
    (*results) = cPlayAndRecResults.rec_result;
    return results;
}
invui::Calls*
VUISessionImpl::all_calls() {
    Synchronized sync(this);
    CheckValid();
    return NULL;
}
void
VUISessionImpl::setProperties(const invui::Properties& props) {
    Synchronized sync(this);
    CheckValid();
    for (int i=0; i<props.length(); i++) {
       setProperty(props[i].name, props[i].value);
    }
}
invui::TelephonyControl_ptr
VUISessionImpl::getTelephonyControl() {
#if 0
    return _telephonyControl;
#else
    return NULL;
#endif
}
invui::VUISession_ptr
```

```
VUISessionImpl::changeOwner(invui::VUIClient_ptr owner) {
#if 0
  return new SessionNode(owner, this);
#else
  return NULL;
#endif
}
void
VUISessionImpl::terminate() {
  Synchronized sync(this);
  CheckValid();
  info("Terminating Session");
  switch (cTelephonyChannel->GetState()) {
        case TelephonyChannel::S_IncomingNotified:
                {
    RejectCallMessage msg(GetCurrentCallToken());
    HandleRejectCall(&msg, NULL);
    break;
    }
        case TelephonyChannel::S_OutgoingDialed:
                {
                RequestToken& token = cTelephonyChannel->GetCurrentRequestToken();
                if (token.IsValid()) {
                        CancelRequestMessage msg(token);
      if (HandleCancelRequest(&msg, NULL)) {
        return;
      }
    }
    }
    // drop call if it failed.
        case TelephonyChannel::S_Connected:
    CDRInfo info;
    DropCallMessage msg(GetCurrentCallToken(), info);
    cTelephonyChannel->HandleDropCall(this, &msg, NULL);
  }
}
invui::RecordResult*
VUISessionImpl::record(const char* filename,
                invui::TIME bos_timeout,
                invui::TIME tms_timeout,
                invui::TIME eos_timeout)
{
  Synchronized sync(this);
  diag("record : filename=%s, bos=%d, tms=%d, eos=%d",
     filename, bos_timeout, tms_timeout, eos_timeout);
  CheckValid();
  CallToken &ct = GetCurrentCallToken();
  RecordMessage msg(ct, filename, bos_timeout, tms_timeout, eos_timeout);
  cRecordMediaChannel->HandleRecord(this, &msg, NULL);
  cRecordingState = RECORDING_STARTED;
  invui::TIME until = tms_timeout + ::GetTickCount();
  invui::TIME timeLeft = tms_timeout;
/*  while (CheckValid() &&
      CheckMedia() &&
      timeLeft>0) {
    sync.wait(timeLeft);
```

```
    timeLeft = until - ::GetTickCount();
  }
*/
  while (CheckValid() &&
      CheckMedia() &&
      cRecordingState != RECORDING_DONE &&
      timeLeft > 0) {
    sync.wait(timeLeft);
    timeLeft = until - ::GetTickCount();
  }
  //StopRecordMessage msg2(ct);
  //cRecordMediaChannel->HandleStop(this, &msg2, NULL);
  if (cRecordingState != RECORDING_DONE)
  {
    StopRecordMessage msg2(ct);
    cRecordMediaChannel->HandleStop(this, &msg2, NULL);
    cRecordResult.reason = invui::RDR_TooMuchSpeech;
    cRecordResult.duration = tms_timeout;
  }
  cRecordingState = RECORDING_NONE;
  invui::RecordResult* result = new invui::RecordResult();
  (*result) = cRecordResult;
  // result->filename = "recorded.wav"; // String_var
  //result->duration = bos_timeout;
  return result;
}
CORBA::Boolean
VUISessionImpl::cancelRecord() {
  return false;
}
CORBA::Boolean
VUISessionImpl::cancelRecognize() {
  return false;
}
//
// HD
//
CORBA::Boolean
VUISessionImpl::cancelPlay() {

  Synchronized sync(this);
  CheckValid();

  return cPlayMediaChannel->DoStop();
}
//
// HD
//
CORBA::Boolean
VUISessionImpl::stopPlay(CORBA::Boolean wait_for_segment) {

  Synchronized sync(this);
  CheckValid();
  diag("stopPlay");
      if(wait_for_segment)
      {
```

```
            cPlayMediaChannel->StopCompositePlay();
            diag("stopPlay: wait for prompts...");

            while (CheckValid() &&
          CheckMedia() &&
          cPromptState == PROMPT_PLAYING) { // until DONE
        sync.wait();
      }

            }
            else cPlayMediaChannel->DoStop();
    cPromptState = PROMPT_NONE;
    diag("stopPlay: done");
            return true;
}
char*
VUISessionImpl::getProperty(const char* key) {
    Synchronized sync(this);
    CheckValid();
    gm::string value;
    cProperties.GetProperty(key, value);
    return CORBA::string_dup(value);
}

void
VUISessionImpl::setProperty(const char* key, const char* value) {
    Synchronized sync(this);
    CheckValid();
    if (strncmp(key, "System.", 7) != 0) {
        Property prop(key, value);
        SetProperty(prop);
    }
}
void
VUISessionImpl::setSearchPath(const char* _path) {
    setProperty("Media.PromptSearchPath", _path);
}
/**
 * Telephony Methods.
 */
CORBA::Boolean
VUISessionImpl::acceptCall(const invui::Call& call){
    Synchronized sync(this);
    diag("acceptCall : id = %d", call.id);
    CheckValid();
    if (cTelephonyChannel->GetState() !=
        TelephonyChannel::S_IncomingNotified) {
        error("AcceptCall in wrong state %d", cTelephonyChannel->GetState());
        // WE NEED ERROR FOR THIS
        throw invui::TelephonyError(TSPF_CALLNOTOFFERED);
    }
    CDRInfo cdrinfo;
    CallToken token = CallTokenMgr::GetCallToken(call.id);
    PropertyList tmp;
    AcceptCallMessage msg(token, &tmp, cdrinfo);
    boolean ret = HandleAcceptCall(&msg, NULL);
```

```
  if (ret == false) {
    return false;
  }
  while (CheckValid() &&
      CheckTelephony() &&
      cTelephonyChannel->GetState() ==
      TelephonyChannel::S_IncomingNotified) {
    sync.wait();
  }
  info("call accepted %d", call.id);
  return cTelephonyChannel->GetState() == TelephonyChannel::S_Connected;
}
void
VUISessionImpl::rejectCall(const invui::Call& call){
  Synchronized sync(this);
  diag("rejectCall : id = %d", call.id);
  CheckValid();
  CallToken token = CallTokenMgr::GetCallToken(call.id);
  RejectCallMessage msg(token);
  HandleRejectCall(&msg, NULL);
}
CORBA::Boolean
VUISessionImpl::cancelCall(const invui::Call& call){
  Synchronized sync(this);
  diag("cancelCall : id = %d", call.id);
  CheckValid();
  RequestToken& token = cTelephonyChannel->GetCurrentRequestToken();
  if (token.IsValid() == false) {
    error("cancelCall: call is not valid");
    throw invui::InvalidState();
  }
  CancelRequestMessage msg(token);
  return HandleCancelRequest(&msg, NULL);
}
invui::Call*
VUISessionImpl::placeCall(const char* number,
                          const char* ani,
                                                  const invui::UUIType& uui,
                                                  invui::TIME timeout)
{
  diag("placeCall : number = %s, ani=%s, timeout=%d",
      number, ani, timeout);
  MTSServerImpl* mtsserver = cSessionMgr->GetMTSServerImpl();
  invui::MTSServer_ops* _ms = static_cast<invui::MTSServer_ops*>(mtsserver);
  return _ms->placeCall(cVUIClient, number, ani, uui, timeout);
}
void
VUISessionImpl::connectCalls(const invui::Call& c1, const invui::Call& c2)
{
  diag("connectCalls : call 1 = %d, call 2 = %d", c1.id, c2.id);
  CallToken token1 = CallTokenMgr::GetCallToken(c1.id);
  CallToken token2 = CallTokenMgr::GetCallToken(c2.id);
  ConnectCallsMessage msg(token1, token2);
  unsigned long ret = GetTSP()->HandleConnectCalls(&msg, NULL);
  switch (ret) {
  case TSPF_OK:
```

```
    return;
  case TSPF_INVALIDCALLTOKEN:
    throw invui::InvalidCall();
  case TSPF_SYSTEMFAILURE:
    throw invui::TelephonyError(TSPF_SYSTEMFAILURE);
  default:
    throw invui::TelephonyError(TSPF_SYSTEMFAILURE);
  }
}
void
VUISessionImpl::disconnectCalls(const invui::Call& c1, const invui::Call& c2)
{
  diag("disconnectCalls : call 1 = %d, call 2 = %d", c1.id, c2.id);
  CallToken token1 = CallTokenMgr::GetCallToken(c1.id);
  CallToken token2 = CallTokenMgr::GetCallToken(c1.id);
  DisconnectCallsMessage msg(token1, token2);
  unsigned long ret = GetTSP()->HandleDisconnectCalls(&msg, NULL);
  switch (ret) {
  case TSPF_OK:
    return;
  case TSPF_INVALIDCALLTOKEN:
    throw invui::InvalidCall();
  case TSPF_SYSTEMFAILURE:
    throw invui::TelephonyError(TSPF_SYSTEMFAILURE);
  default:
    throw invui::TelephonyError(TSPF_SYSTEMFAILURE);
  }
}
/**
 * Logging Methods
 */
void
VUISessionImpl::enterState(const char* _name) {
  Synchronized sync(this);
  CheckValid();
  cSessionLogger.NotifyEnteringState(_name);
}
void
VUISessionImpl::leaveState() {
  Synchronized sync(this);
  CheckValid();
  cSessionLogger.NotifyLeaveState();
}
void
VUISessionImpl::log(const char* _name, const char* _value) {
  Synchronized sync(this);
  CheckValid();
  cSessionLogger.LogItem(_name, _value);
}
invui::Call*
VUISessionImpl::transfer(const char* _number, invui::TIME timeout) {
  CheckValid();
  return NULL;
}
void
VUISessionImpl::dropCalls() {
```

```
    Synchronized sync(this);
    diag("dropCalls");
    CheckValid();
}
void
VUISessionImpl::dropCall(const invui::Call& c1) {
    CheckValid();
}
```

Document ID: 2.0.0.0

```
/*
 * WFStreamer.cpp
 *
 * Copyright (c) 2000 - General Magic, Inc. - All rights reserved
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF GENERAL MAGIC
 * Use and reproduction restricted by U.S. copyright law and
 * your contract with General Magic.
 *
 * $Revision: 1.26 $
 * $Date: 2000/12/07 03:19:57 $
 */
static const char* id="$Id: WFStreamer.cpp,v 1.26 2000/12/07 03:19:57 Mitsuru Oshima Exp $";
#include "WFStreamer.h"
#include "VPPUtils/VPPDebug.h"
#include "SampleListener.h"
#include "SessionNode.h"
typedef unsigned (__stdcall *PBEGINTHREADEX_THREADFUNC) (
                                        LPVOID lpThreadParameter
                                    );
typedef unsigned *PBEGINTHREADEX_THREADID;
WFStreamer::WFStreamer(const char *filename, int bufSize, int transmissionInterval)
: cBufSize(bufSize), cTransIntervalmsecs(transmissionInterval),
  cpFile(NULL), cpFilename(NULL),
  cpBuf(NULL), cpSampleListener(NULL),
  cHeaderRead(false),
  cStatus(eS_OK), cState(eS_Idle),
  cThreadH(NULL), cFileWriteStopped(false),
  cDeleteFile(true), cNumDeleteTries(0)
#ifdef ENABLE_PERF_MEASURE
  , cFirstSample(true), cPushSamplesTime(0L)
#endif
{
        if (filename) {
                cpFilename = new char[strlen(filename) + 1];
                strcpy(cpFilename, filename);
        }
        else {
                cStatus = eS_NotOK;
                cpFilename = new char[1];
                cpFilename[0] = '\0';
                debug::printf("WFStreamer", "WFStreamer %p, Status Not OK.\n", this); // go past the header
        }
        cpBuf = new char[cBufSize];
        cStopTransmissionEvent = ::CreateEvent(NULL, true, false, NULL);  // must be manually reset!
}
WFStreamer::~WFStreamer(void)
{
        delete [] cpBuf;
        delete [] cpFilename;
        ::CloseHandle(cStopTransmissionEvent);
}
void WFStreamer::Read(void)
{
        size_t numCharsRead;
        if (!cHeaderRead) {
```

```
                while (fseek(cpFile, 0x2C, SEEK_SET) != 0) {
                        debug::printf("WFStreamer", "Not past header yet\n"); // go past the header
                }
                fgetpos(cpFile, &cCurFilePos);
                cHeaderRead = true;
        }
        // read the wavefile
        fsetpos(cpFile, &cCurFilePos);
        if (!feof(cpFile)) {
                numCharsRead = fread(cpBuf, sizeof(char), cBufSize, cpFile);
                if (ferror(cpFile)) {
                        perror( "Some read error occurred" );
                }
                if (numCharsRead > 0) {
                        // notify listener
                        if (cpSampleListener) {
#ifdef ENABLE_PERF_MEASURE
                                // clock the start of samples transmitting
                                if (cFirstSample)
                                {
                                        cFirstSample = false;
                                        unsigned long tSendSamples = ::GetTickCount();
                                        cpSampleListener->SetSamplesTime(tSendSamples);
                                        cPushSamplesTime = tSendSamples;
                                        debug::printf("PerfMeasure", "Time it took to start sampling: %ld msecs,
chars read %d\n", tSendSamples - cStartRecordingTime, numCharsRead);
                                }
                                else
                                {
                                        unsigned long time = ::GetTickCount();
                                        cPushSamplesTime = time;
                                        debug::printf("PerfMeasure", "It took %ld msecs between streamer
pushsamples, chars read %d.\n", time - cPushSamplesTime, numCharsRead);
                                }
#endif
                                cpSampleListener->NotifySample((const unsigned char *)cpBuf, numCharsRead);
                        }
                        fgetpos(cpFile, &cCurFilePos);
                }
        }
        return;
}
FILE *
WFStreamer::OpenFile(const char *filename, const char *mode)
{
        FILE *pFile = NULL;
        if ((pFile = fopen(filename, mode)) == NULL) {
                debug::printf("WFStreamer", "Streamer Thread: cannot open file %s\n", filename);
        }
        else
                debug::printf("WFStreamer", "Streamer Thread: file %s opened successfully\n", filename);
        return pFile;
}
HANDLE
WFStreamer::Go(void)
{
```

```
        cThreadH = (HANDLE)_beginthreadex( NULL,
                0,
                (PBEGINTHREADEX_THREADFUNC) goThreadFunc,
                (LPVOID) this,
                0,
                (PBEGINTHREADEX_THREADID) &cThreadId );
        if (cThreadH) {
                debug::printf("WFStreamer", "Streamer Thread launched\n");
        }
        return cThreadH;
}
unsigned __stdcall
WFStreamer::goThreadFunc(LPVOID param)
{
        WFStreamer *pThisObj = (WFStreamer *)param;
        return pThisObj->GoThreadFunc();
}
unsigned
WFStreamer::GoThreadFunc(void)
{
        while ((cpFile = OpenFile(cpFilename, "rb")) == NULL )
                debug::printf("WFStreamer", "Waiting for file %s\n", cpFilename);
        unsigned long waittime = cTransIntervalmsecs ? cTransIntervalmsecs : 3L;
        cDeleteFile = true;
        cCurFilePos = 0;
        while (true) {
                Read();
                if (cFileWriteStopped) {
                        fseek(cpFile, 0L, SEEK_END);
                        fgetpos(cpFile, &cEOFPos);
                        if (cCurFilePos == cEOFPos)
                                break;
                }
                // sleep for cTransIntervalmsecs milliseconds but break if requested to stop
                if (::WaitForSingleObject(cStopTransmissionEvent, waittime) != WAIT_TIMEOUT) { // signaled
                        debug::printf("WFStreamer", "Stop Transmission is requested during the \"sleep\"
period.\n");
                        ::ResetEvent(cStopTransmissionEvent);
                        break;
                }
        }
        fclose(cpFile);     // close file before calling reset
        ::CloseHandle(cThreadH);
        Reset();
        return 0;
}
void
WFStreamer::NotifyFileWriteStopped(void)
{
        debug::printf("WFStreamer", "WFStreamer Thread: File writing is stopped\n");
        cFileWriteStopped = true;
        return;
}
bool
WFStreamer::StartTransmission(void)
{
```

```
        if (IsReadyToTransmit()) {
                cState = eS_Streaming;
                cFileWriteStopped = false;
                ::ResetEvent(cStopTransmissionEvent);
                Go();
                return true; // to do
        }
        else {
                debug::printf("WFStreamer", "WFStreamer %p, Not ready to transmit.\n", this); // go past the header
                return false;
        }
}
bool
WFStreamer::StopTransmission(void)
{
        if (cState == eS_Streaming) {
                ::SetEvent(cStopTransmissionEvent); // stop is requested - stop thread
                return true;
        }
        else
                return false;
}
bool
WFStreamer::IsReadyToTransmit(void)
{
        debug::printf("WFStreamer", "WFStreamer %p, cStatus %d, cState %d, cpSampleListener %p.\n", this,
cStatus, cState, cpSampleListener); // go past the header
        return (cStatus == eS_OK &&
                        cState  == eS_Idle &&
                                cpSampleListener);
}
void
WFStreamer::Reset(void)
{
        cFileWriteStopped = false;
        cpSampleListener  = NULL;
        cThreadH    = NULL;
        cHeaderRead = false;
        cpFile      = NULL;
        cNumDeleteTries = 0;
        ::ResetEvent(cStopTransmissionEvent);
}
// return value bool is reserved for future use
//
bool
WFStreamer::WaitTillIdle(void)
{
        if (cState != eS_Idle) {
                ::WaitForSingleObject(cThreadH, INFINITE);
                //::CloseHandle(cThreadH);
                cState = eS_Idle;
        }
// debug - temporarily not deleting files
//      return true;
        // clean up file
        if (cDeleteFile) {
```

```
                if (!_unlink(cpFilename) || (cNumDeleteTries > 15)) {// successful deletion
                        cDeleteFile = false;
                        cNumDeleteTries = 0;
                        debug::printf("WFStreamer", "Streamer Thread: file %s is deleted.\n", cpFilename);
                        return true;
                }
                else {
                        cNumDeleteTries++;
                        debug::printf("WFStreamer", "Streamer Thread: cannot delete file %s.\n", cpFilename);
                        return false;
                }
        }
        else
                return true;
}
void
WFStreamer::SetDeleteFile(bool bDelete)
{
        cDeleteFile = bDelete;
}
```

```
package com.genmagic.invui.vxml.interp;
import org.w3c.dom.Document;
import org.w3c.dom.DOMException;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import com.genmagic.invui.vxml.dom.Catch;
import com.genmagic.invui.vxml.dom.Vxml;
import com.genmagic.invui.util.Log;
import com.genmagic.invui.util.LogFactory;
import java.util.Vector;
/**
 * An AppContext class is responsible for managing
 * application state such as both application and current
 * document, the source url of those documents, etc.
 */
public class AppContext implements VXMLConstants {
    private XMLLoader appLoader = null;
    private XMLLoader currentLoader = null;
    private XMLLoader rollbackLoader = null;
    private ResourceManager resourceManager = null;
    private Catch[] eventHandlers = null;
    private Properties properties = null;
    static protected Log _log = LogFactory.getLogFor("vml");
    public AppContext(XMLLoader l, ResourceManager rm) {
        currentLoader = l;
        resourceManager = rm;
    }
    public XMLLoader getLoader() {
        return currentLoader;
    }
    public XMLLoader getApplicationRootLoader() {
        return appLoader;
    }
    public ResourceManager getResourceManager() {
        return resourceManager;
    }
    public void setApplicationRootPath(String url) throws java.net.MalformedURLException {
        if (appLoader == null) {
            appLoader = (XMLLoader)currentLoader.cloneLoader();
        }
        appLoader.setPath(url);
    }
    public boolean isInSameApplicationRoot(String path) throws java.net.MalformedURLException {
        Loader newLoader = currentLoader.createLoader(path);
        return newLoader.sameURL(appLoader);
    }
    public void clearApplicationRoot() {
        appLoader = null;
    }
    public void setCurrentAsApplicationRoot() {
        appLoader = (XMLLoader)currentLoader.cloneLoader();
    }
    public void setApplicationRootAsCurrent() {
        currentLoader = (XMLLoader)appLoader.cloneLoader();
        rollbackLoader = (XMLLoader)currentLoader.cloneLoader();
    }
```

```
public boolean isInApplicationRoot() {
   return currentLoader.sameURL(appLoader);
}
public void loadDocument(int method) throws EventException {
   loadDocument(method, 0, null);
}
public void loadDocument(int method, int timeout, String enctype) throws EventException {
   try {
      currentLoader.loadDocument(method, timeout, enctype);
      // to rollback to the original document
      rollbackLoader = (XMLLoader)currentLoader.cloneLoader();
   } catch (java.io.InterruptedIOException ex) {
      String path = currentLoader.getPath();
      if (rollbackLoader != null) {
         currentLoader = rollbackLoader;
      }
      _log.error("io exception : timeout in loading document : "+
            path + " , msg="+
            ex.getMessage());

      throw new EventException("error.badfetch.io.timeout",
                        "cannot load document " +
                        path +
                        " due to " + ex.getMessage(),
                        ex);
   } catch (java.io.IOException ex) {
      String path = currentLoader.getPath();
      if (rollbackLoader != null) {
         currentLoader = rollbackLoader;
      }
      _log.error("io exception in loading document : "+
            path + " , msg="+
            ex.getMessage());

      throw new EventException("error.badfetch.io",
                        "cannot load document " +
                        path +
                        " due to " + ex.getMessage(),
                        ex);
   } catch (SAXParseException ex) {
      String path = currentLoader.getPath();
      if (rollbackLoader != null) {
         currentLoader = rollbackLoader;
      }
      _log.error("sax parse exception in loading document : "+
            path + " , msg="+
            ex.getMessage());

      throw new EventException("error.badfetch.sax",
                        "could not parse loaded document " +
                        path +
                        " due to " + ex.getMessage(),
                        ex);
   } catch (SAXException ex) {
      String path = currentLoader.getPath();
      if (rollbackLoader != null) {
```

```
            currentLoader = rollbackLoader;
          }
          _log.error("sax exception in loading document : "+
                  path + " , msg="+
                  ex.getMessage());
          throw new EventException("error.badfetch.sax",
                          "cannot load document " +
                          path +
                          " due to " + ex.getMessage(),
                          ex);
        } catch (DOMException ex) {
          String path = currentLoader.getPath();
          if (rollbackLoader != null) {
            currentLoader = rollbackLoader;
          }
          _log.error("dom exception in loading document : "+
                  path + " , msg="+
                  ex.getMessage());
          throw new EventException("error.badfetch.dom",
                          "cannot load document " +
                          path +
                          " due to " + ex.getMessage(),
                          ex);
        } catch (RuntimeException ex) {
          if (rollbackLoader != null) {
            currentLoader = rollbackLoader;
          }
          throw ex;
        } catch (Error ex) {
          if (rollbackLoader != null) {
            currentLoader = rollbackLoader;
          }
          throw ex;
        }
      }
      public Document getDocument() {
        return currentLoader.getDocument();
      }
      public Document getApplicationRootDocument() {
        if (appLoader != null) {
          return appLoader.getDocument();
        } else {
          return null;
        }
      }
      public void getApplicationGrammars(VXMLInterp ip, GrammarSet set) throws EventException {
        if (appLoader != null) {
          Vxml vxml = (Vxml)getApplicationRootDocument().getDocumentElement();
          vxml.getDocumentGrammars(ip, set);
        }
        // return grammars;
      }
    /*
      public Properties getApplicationProperties() {
        if (properties == null && appLoader != null) {
          Vxml vxml = (Vxml)getApplicationRootDocument().getDocumentElement();
```

```
        properties = vxml.getProperties();
      }
      return properties;
    }
  */
  /*
    private Catch[] getEventHandlers() {
      if (eventHandlers == null) {
        eventHandlers = new Catch[0];
      }
      return eventHandlers;
    }
    public void collectEventHandlers(Vector v, String eventName) {
      Catch[] handlers = getEventHandlers();
      for (int i=0; i<handlers.length; i++) {
        Catch c = handlers[i];
        if (eventName.equals(c.getEventName())) {
          EventHandlerSet.addEventHandler(v, c);
        }
      }
    }
  */
    public void loadApplicationRoot() throws EventException {
      properties = null;
      eventHandlers = null;
      try {
        appLoader.loadDocument(GET);
      } catch (java.io.InterruptedIOException ex) {
        _log.error("io exception in loading root document : "+
              appLoader.getPath() + " , msg="+
              ex.getMessage());
        throw new EventException("error.semantic.badapp.timeout",
                    "cannot load application root " +
                    appLoader.getPath() +
                    " due to " + ex.getMessage(),
                    ex);
      } catch (java.io.IOException ex) {
        _log.error("io exception in loading root document : "+
              appLoader.getPath() + " , msg="+
              ex.getMessage());
        throw new EventException("error.semantic.badapp",
                    "cannot load application root " +
                    appLoader.getPath() +
                    " due to " + ex.getMessage(),
                    ex);
      } catch (SAXParseException ex) {
        _log.error("sax parse exception in loading root document : "+
              appLoader.getPath() + " , msg="+
              ex.getMessage());
        throw new EventException("error.semantic.badapp.sax",
                    "could not parse application root document" +
                    appLoader.getPath() +
                    " due to " + ex.getMessage(),
                    ex);
      } catch (SAXException ex) {
        _log.error("sax exception in loading root document : "+
```

```
                appLoader.getPath() + " , msg="+
                ex.getMessage());
            throw new EventException("error.semantic.badapp.sax",
                        "cannot load application root " +
                        appLoader.getPath() +
                        " due to " + ex.getMessage(),
                        ex);
        } catch (DOMException ex) {
            _log.error("dom exception in loading root document : "+
                appLoader.getPath() + " , msg="+
                ex.getMessage());
            throw new EventException("error.semantic.badapp.dom",
                        "cannot load application root " +
                        appLoader.getPath() +
                        " due to " + ex.getMessage(),
                        ex);
        }
    }
    public void initApplicationRoot(VXMLInterp ip) throws VXMLException {
        eventHandlers = null;
        Document doc = getApplicationRootDocument();
        Vxml vxml = (Vxml)doc.getDocumentElement();
        if (vxml.getApplication() != null) {
            throw new EventException("error.semantic",
                        "root document cannot have root document");
        }
        ip.beginApplicationScope();
        ip.beginPropertyScope();
        ip.eval(vxml);
        // ip.eval(vxml.getVarDecls());
    }
    public void shutdown() {
        if (currentLoader != null) {
            currentLoader.close();
        }
        if (appLoader != null) {
            appLoader.close();
        }
    }
}
```

```
package com.genmagic.invui.vxml.interp;
public class AssertException extends RuntimeException {
   Throwable t = null;
   public AssertException(String msg, Throwable t) {
      super(msg);
      t = t;
   }
   public AssertException(String msg) {
      super(msg);
   }
   public String getMessage() {
      if (t == null) {
         return super.getMessage();
      } else {
         return super.getMessage() + " exception = " + t.getMessage();
      }
   }
}
```

```
package com.genmagic.invui.vxml.dom;
import com.genmagic.invui.vxml.interp.EventException;
abstract public class Assign extends VXMLElement {
   public Assign() {
      super("assign");
   }
   /**
    * Get expression.
    */
   abstract public Expr getExpr() throws EventException;
   /**
    * Get variable name
    */
   abstract public String getName();
}
```

```
package com.genmagic.invui.vxml.tags;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.dom.Assign;
import com.genmagic.invui.vxml.interp.TagHandler;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLException;
public class AssignTag implements TagHandler {
   public void perform(VXMLInterp ip, Element self) throws VXMLException {
      Assign assign = (Assign)self;
      ip.setVariable(assign.getName(),
                  ip.evalExpression(assign.getExpr()));
   }
}
```

Docket No. 2000-0014

```
package com.genmagic.invui.vxml.dom;
import com.genmagic.invui.vxml.interp.EventException;
abstract public class Audio extends VXMLElement implements Fetchable {
   public Audio() {
      super("audio");
   }
   /**
    * Get src url
    */
   abstract public String getSrc();
   /**
    * Get src expression. GMI extension
    *
    */
   abstract public Expr getExprSrc() throws EventException;
   /**
    * Get src url
    * @deprecated as of Vxml1.0
    */
   abstract public String getMode();
}
```

```
package com.genmagic.invui.vxml.tags;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.prompt.CompositePrompt;
import com.genmagic.invui.vxml.dom.Audio;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.interp.*;
public class AudioTag implements TagHandler {
   public void perform(VXMLInterp ip, Element self) throws VXMLException {
      Audio audio = (Audio)self;
      String src = audio.getSrc();
      if (src == null) {
         Expr expr = audio.getExprSrc();
         if (expr != null) {
            src = ip.evalExpression(expr).toString();
         }
         if (src == null) {
            throw new EventException("error.semantic",
                              "no src in audio ");
         }
      }
      String alt = "";
      String content = null;
      SpeechControl speech = ip.getSpeechControl();
      // may needs to be optimized
      if (audio.hasChildNodes()) {
         CompositePrompt backup = speech.getCompositePrompt();
         // Prompt[] backup = ip.getPrompts();
         speech.clearPrompts();
         ip.evalChildren(self);
         // Prompt[] p = ip.getPrompts();
         //ip.clearPrompts();
         // ip.appendPrompts(backup);
         CompositePrompt p = speech.getCompositePrompt();
         speech.clearPrompts();
         backup.appendTo(speech);
         content = p.getAllTTS();
      }
      alt = content;
      src = ip.loadAudio(src,
                  audio.getCaching(),
                  audio.getFetchTimeout());
      speech.appendAudio(src, alt);
   }
}
```

```
package com.genmagic.invui.vxml.property;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
public class BargeinPropertyListener implements PropertyChangeListener {
   public void propertyAdded(VXMLInterp ip, Object value) throws EventException {
      //System.out.println("bargein set " + value);
      if (value instanceof Boolean) {
         ip.setBargein(((Boolean)value).booleanValue());
      } else {
         String name = value.toString();
         if ("true".equals(name)) {
            ip.setBargein(true);
         } else if ("false".equals(name)) {
            ip.setBargein(false);
         } else {
            throw new EventException("error.semantic",
                              "bargein property needs to be boolean: " + value);
         }
      }
   }
   public void propertyRemoved(VXMLInterp ip) {
      boolean origin =
         ip.getCurrentProperties().getBooleanProperty("bargein", true);
      //System.out.println("bargein removed " + origin);
      ip.setBargein(origin);
   }
};
```

```
package com.genmagic.invui.vxml.script.expr;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
/** interface for binary function in expressions
 * @see Expression
 */
interface BinaryFunction extends Function {
   Object perform(VXMLInterp ip, Object arg1, Object arg2) throws EventException;
}
```

Document Page

```
package com.genmagic.invui.vxml.dom;
abstract public class Block extends FormItem {
   public Block() {
      super("block");
   }
}
```

Document ID: 2

```
package com.genmagic.invui.vxml.tags;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.dom.Block;
import com.genmagic.invui.vxml.interp.*;
public class BlockTag extends ExecutableTag {
   public void perform(VXMLInterp ip, Element self) throws VXMLException {
      Block block = (Block)self;
      // See VoiceXML1.0 specification 14.2 Block page 54
      ip.setVariable(block.getName(), Boolean.TRUE);
      super.perform(ip, self);
   }
}
```

```
package com.genmagic.invui.vxml.dom;
abstract public class Break extends VXMLElement {
   static public final short NONE = -1;
   static public final short SMALL = 0;
   static public final short MEDIUM = 1;
   static public final short LARGE = 2;
   public Break() {
      super("break");
   }
   abstract public short getSize();
   /*
    * @deprecated as of Vxml1.0
    */
   abstract public int getLength();
   /*
    * @since Vxml1.0
    */
   abstract public int getMsecs();
}
```

Document Retrieved from Patent Genius

```
package com.genmagic.invui.vxml.tags;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.dom.Break;
import com.genmagic.invui.vxml.interp.*;
public class BreakTag implements TagHandler {
   public void perform(VXMLInterp ip, Element self) throws VXMLException {
      Break brk = (Break)self;
      ip.getSpeechControl().appendSilence(brk.getMsecs());
   }
}
```

Document Page 30

```
package com.genmagic.invui.vxml.script.expr;
import java.util.Hashtable;
import java.text.*;
import org.w3c.dom.*;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.lex.*;
class BuiltInFunctions {
   static public void init(Hashtable keywords) {
      keywords.put("Math.abs",  new AbsFunction());
      keywords.put("Math.sin",  new SinFunction());
      keywords.put("Math.cos",  new CosFunction());
      keywords.put("Math.tan",  new TanFunction());
      keywords.put("Math.asin", new AsinFunction());
      keywords.put("Math.acos", new AcosFunction());
      keywords.put("Math.atan", new AtanFunction());
      keywords.put("Math.atan2",new Atan2Function());
      keywords.put("Math.exp",  new ExpFunction());
      keywords.put("Math.log",  new LogFunction());
      keywords.put("Math.sqrt", new SqrtFunction());
      keywords.put("Math.ceil", new CeilFunction());
      keywords.put("Math.floor",new FloorFunction());
      keywords.put("Math.rint", new RintFunction());
      keywords.put("Math.round",new RoundFunction());
      keywords.put("Math.random", new RandFunction());
      keywords.put("Math.int",  new IntFunction());
      keywords.put("Math.float",new FloatFunction());
      keywords.put("Math.pow", new PowFunction());
      keywords.put("Math.max", new MaxFunction());
      keywords.put("Math.min", new MinFunction());
      keywords.put("Time.current", new TimeCurrentFunction());
      keywords.put("Runtime.freeMemory", new FreeMemFunction());
      keywords.put("Runtime.totalMemory", new TotalMemFunction());
      /*
        keywords.put("abs",  new AbsFunction());
        keywords.put("sin",  new SinFunction());
        keywords.put("cos",  new CosFunction());
        keywords.put("tan",  new TanFunction());
        keywords.put("asin", new AsinFunction());
        keywords.put("acos", new AcosFunction());
        keywords.put("atan", new AtanFunction());
        keywords.put("atan2",new Atan2Function());
        keywords.put("exp",  new ExpFunction());
        keywords.put("log",  new LogFunction());
        keywords.put("sqrt", new SqrtFunction());
        keywords.put("ceil", new CeilFunction());
        keywords.put("floor",new FloorFunction());
        keywords.put("rint", new RintFunction());
        keywords.put("round",new RoundFunction());
        keywords.put("random", new RandFunction());
        keywords.put("int",  new IntFunction());
        keywords.put("float",new FloatFunction());
        keywords.put("pow", new PowFunction());
        keywords.put("max", new MaxFunction());
        keywords.put("min", new MinFunction());
        // STRING
```

```
        keywords.put("trim", new TrimFunction());
        keywords.put("index", new IndexFunction());
        keywords.put("substring", new SubstringFunction());
        // converter
        keywords.put("date",       new DateFunction());
        keywords.put("dateformat", new DateFormatFunction());
        keywords.put("number",     new NumberFunction());
        //
        keywords.put("exists",   new ExistsFunction());
        // keywords.put("call",    new CallFunction());
        */
    }
}
class TimeCurrentFunction implements NoArgFunction {
    public Object perform(VXMLInterp ip) throws ExpressionException {
        return new Long( System.currentTimeMillis() );
    }
}
class FreeMemFunction implements NoArgFunction {
    public Object perform(VXMLInterp ip) throws ExpressionException {
        return new Long( Runtime.getRuntime().freeMemory() );
    }
}
class TotalMemFunction implements NoArgFunction {
    public Object perform(VXMLInterp ip) throws ExpressionException {
        return new Long( Runtime.getRuntime().totalMemory() );
    }
}
class RandFunction implements NoArgFunction {
    public Object perform(VXMLInterp ip) throws ExpressionException {
        return new Double( Math.random() );
    }
}
class SinFunction implements UnaryFunction {
    public Object perform(VXMLInterp ip, Object arg) throws ExpressionException {
        return new Double(Math.sin(Expression.toNumber(arg).doubleValue()));
    }
}
class CosFunction implements UnaryFunction {
    public Object perform(VXMLInterp ip, Object arg) throws ExpressionException {
        return new Double(Math.cos(Expression.toNumber(arg).doubleValue()));
    }
}
class TanFunction implements UnaryFunction {
    public Object perform(VXMLInterp ip, Object arg) throws ExpressionException {
        return new Double(Math.tan(Expression.toNumber(arg).doubleValue()));
    }
}
class AsinFunction implements UnaryFunction {
    public Object perform(VXMLInterp ip, Object arg) throws ExpressionException {
        return new Double(Math.asin(Expression.toNumber(arg).doubleValue()));
    }
}
class AcosFunction implements UnaryFunction {
    public Object perform(VXMLInterp ip, Object arg) throws ExpressionException {
        return new Double(Math.cos(Expression.toNumber(arg).doubleValue()));
```

```
    }
}
class AtanFunction implements UnaryFunction {
    public Object perform(VXMLInterp ip, Object arg) throws ExpressionException {
        return new Double(Math.atan(Expression.toNumber(arg).doubleValue()));
    }
}
class AbsFunction implements UnaryFunction {
    public Object perform(VXMLInterp ip, Object arg) throws ExpressionException {
        if (arg instanceof Double) {
            return new Double(Math.abs( ((Number)arg).doubleValue() ));
        } else {
            return new Long(Math.abs(Expression.toNumber(arg).longValue()));
        }
    }
}
class ExpFunction implements UnaryFunction {
    public Object perform(VXMLInterp ip, Object arg) throws ExpressionException {
        return new Double(Math.exp(Expression.toNumber(arg).doubleValue()));
    }
}
class LogFunction implements UnaryFunction {
    public Object perform(VXMLInterp ip, Object arg) throws ExpressionException {
        return new Double(Math.log(Expression.toNumber(arg).doubleValue()));
    }
}
class SqrtFunction implements UnaryFunction {
    public Object perform(VXMLInterp ip, Object arg) throws ExpressionException {
        return new Double(Math.sqrt(Expression.toNumber(arg).doubleValue()));
    }
}
class IntFunction implements UnaryFunction {
    public Object perform(VXMLInterp ip, Object arg) throws ExpressionException {
        return new Long( Expression.toNumber(arg).longValue() );
    }
}
class FloatFunction implements UnaryFunction {
    public Object perform(VXMLInterp ip, Object arg) throws ExpressionException {
        return new Double( Expression.toNumber(arg).doubleValue() );
    }
}
class CeilFunction implements UnaryFunction {
    public Object perform(VXMLInterp ip, Object arg) throws ExpressionException {
        return new Double( Math.ceil(Expression.toNumber(arg).doubleValue()));
    }
}
class FloorFunction implements UnaryFunction {
    public Object perform(VXMLInterp ip, Object arg) throws ExpressionException {
        return new Double( Math.floor(Expression.toNumber(arg).doubleValue()));
    }
}
class RintFunction implements UnaryFunction {
    public Object perform(VXMLInterp ip, Object arg) throws ExpressionException {
        return new Double( Math.rint(Expression.toNumber(arg).doubleValue()));
    }
}
```

```
class RoundFunction implements UnaryFunction {
  public Object perform(VXMLInterp ip, Object arg) throws ExpressionException {
    return new Long( Math.round(Expression.toNumber(arg).doubleValue()));
  }
}
class TrimFunction implements UnaryFunction {
  public Object perform(VXMLInterp ip, Object arg) throws ExpressionException {
    return arg.toString().trim();
  }
}
class Atan2Function implements BinaryFunction {
  public Object perform(VXMLInterp ip, Object arg1, Object arg2) throws ExpressionException {
    return new Double(Math.atan2(Expression.toNumber(arg1).doubleValue(),
                    Expression.toNumber(arg2).doubleValue()));
  }
}
class PowFunction implements BinaryFunction {
  public Object perform(VXMLInterp ip, Object arg1, Object arg2) throws ExpressionException {
    return new Double(Math.pow(Expression.toNumber(arg1).doubleValue(),
                    Expression.toNumber(arg2).doubleValue()));
  }
}
class MaxFunction implements BinaryFunction {
  public Object perform(VXMLInterp ip, Object arg1, Object arg2) throws ExpressionException {
    int t = Expression.mathType(arg1, arg2);
    if (t == INTEGER) {
      return new Long(Math.max(((Number)arg1).longValue(),
                    ((Number)arg2).longValue()));
    } else if (t == FLOAT) {
      return new Double(Math.max(((Number)arg1).doubleValue(),
                    ((Number)arg2).doubleValue()));
    } else {
      throw new ExpressionException("Number arguments are expected in max()");
    }
  }
}
class MinFunction implements BinaryFunction {
  public Object perform(VXMLInterp ip, Object arg1, Object arg2) throws ExpressionException {
    int t = Expression.mathType(arg1, arg2);
    if (t == INTEGER) {
      return new Long(Math.min(((Number)arg1).longValue(),
                    ((Number)arg2).longValue()));
    } else if (t == FLOAT) {
      return new Double(Math.min(((Number)arg1).doubleValue(),
                    ((Number)arg2).doubleValue()));
    } else {
      throw new ExpressionException("Number arguments are expected in min()");
    }
  }
}
class IndexFunction implements BinaryFunction {
  public Object perform(VXMLInterp ip, Object arg1, Object arg2) throws ExpressionException {
    // type check?
    return new Long(arg1.toString().indexOf(arg2.toString()));
  }
}
```

```
class SubstringFunction implements NargFunction {
  public Object perform(VXMLInterp ip, Object[] args) throws ExpressionException {
    // type check?
    if ((args.length != 2) && (args.length != 3)) {
      throw new ExpressionException("Wrong number arguments in substring " + args.length);
    }
    String str = args[0].toString();
    int start, end;
    if (args[1] instanceof String) {
      String a1 = args[1].toString();
      start = str.indexOf(a1);
      if (start <0) {
        throw new ExpressionException("'" + args[1] +
                                      "' not found in " + str);
      }
      start += a1.toString().length();
    } else {
      start = Expression.toNumber(args[1]).intValue();
    }
    if (args.length == 2) {
      end = str.length();
    } else if (args[2] instanceof String) {
      end = str.indexOf(args[2].toString(), start);
      if (end <0) {
        throw new ExpressionException("'" + args[2] +
                                      "' not found in " + str);
      }
    } else {
      end = start + Expression.toNumber(args[2]).intValue();
    }
    try {
      return str.substring(start, end);
    } catch (StringIndexOutOfBoundsException ex) {
      throw new ExpressionException("error occured in substring. " + ex.getMessage());
    }
  }
}
class DateFunction implements NargFunction {
  public Object perform(VXMLInterp ip, Object args[]) throws ExpressionException {
    if (args.length > 2) {
      throw new ExpressionException("wrong number arguments in date()");
    }
    String date = args[0].toString();
    String format = "M/d/yy h:mm a";
    if (args.length == 2) {
      format = args[1].toString();
    }
    try {
      DateFormat parser = new SimpleDateFormat(format);
      java.util.Date now = parser.parse(date);
      if (now.getYear() == 70 &&
        (format.indexOf('y') < 0 && format.indexOf('Y') <0)) {
        // Ad hoc way to check
        System.out.println("adjusting year...");
        now.setYear( new java.util.Date().getYear() );
      }
```

```
        return new Double(now.getTime());
      } catch (ParseException ex) {
        throw new ExpressionException("error occured while parsing the date " +ex.getMessage() );
      }
    }
  }
  class DateFormatFunction implements NargFunction {
    public Object perform(VXMLInterp ip, Object args[]) throws ExpressionException {
      if (args.length > 2) {
        throw new ExpressionException("wrong number arguments in date_format()");
      }
      long date = Expression.toNumber(args[0]).longValue();
      String format = "M/d/yy h:mm a";
      if (args.length == 2) {
        format = args[1].toString();
      }
      DateFormat formatter = new SimpleDateFormat(format);
      return formatter.format(new java.util.Date(date));
    }
  }
  class NumberFunction implements NargFunction {
    public Object perform(VXMLInterp ip, Object args[]) throws ExpressionException {
      if (args.length < 1) {
        throw new ExpressionException("wrong number arguments in number()");
      }
      String numberStr = args[0].toString();
      try {
        NumberFormat parser = null;
        if (args.length == 2) {
          String pattern = args[1].toString();
          parser = new DecimalFormat(pattern);
        } else {
          parser = NumberFormat.getInstance();
        }
        return parser.parse(numberStr);
      } catch (ParseException ex) {
        throw new ExpressionException("error occured while parsing the number " +ex.getMessage() );
      }
    }
  }
  class ExistsFunction implements UnaryFunction {
    public Object perform(VXMLInterp ip, Object arg) throws EventException {
      Object obj = ip.getVariable(arg.toString());
      if (obj != null) {
        return new Boolean(true);
      } else {
        return new Boolean(false);
      }
    }
  }
```

```
package com.genmagic.invui.vxml.interp;
import java.util.Hashtable;
import com.genmagic.invui.util.LogFactory;
/**
 * BuiltInGrammars
 */
public class BuiltInGrammar extends Grammar {
   public BuiltInGrammar(String n) {
      _name = n;
   }
   public BuiltInGrammar(String n, float w) {
      _name = n;
      _weight = w;
   }
   public BuiltInGrammar(String n, String[] s) {
      _name = n;
      _slots = s;
   }
   public BuiltInGrammar(String n, String[] s, float w) {
      _name = n;
      _slots = s;
      _weight = w;
   }
   public String getName() {
      return _name;
   }
   public String getUniqID() {
      return getName();
   }
   public String[] getSlotNames() {
      return _slots;
   }
   public float getWeight() {
      return _weight;
   }
   private String _name = null;
   private String[] _slots = null;
   private float _weight = -1.0f;
   public SlotResult getSlotResult(RecResult result, String name) throws EventException {
      if (_slots != null) { // predefined
         return result.getRawResult(_slots[0]);
      } else {
         return result.getRawResult(name);
      }
   }
   static class LinkGrammar extends BuiltInGrammar {
      String event = null;
      LinkGrammar(String name, String ev) {
         super(name, null);
         event = ev;
      }
      public SlotResult getSlotResult(RecResult result, String name) throws EventException {
         throw new EventException(event);
      }
      public void performAction(RecResult result, Object cxt) throws Exception {
         throw new EventException(event);
```

```
    }
  }
  public String toString() {
    return "Builtin:" + super.toString();
  }
  static class BooleanGrammar extends BuiltInGrammar {
    BooleanGrammar() {
      super("D_YesNo", new String[]{"boolean"});
    }
    public SlotResult getSlotResult(RecResult result, String name) throws EventException {
      SlotResult r = result.getRawResult("boolean");
      if (r != null && "yes".equals(r.value)) {
        r.value = Boolean.TRUE;
      } else {
        r.value = Boolean.FALSE;
      }
      return r;
    }
  };
  static class MoneyGrammar extends BuiltInGrammar {
    MoneyGrammar() {
      super("D_Money", new String[]{"money"});
    }
    public SlotResult getSlotResult(RecResult result, String name) throws EventException {
      SlotResult dollars = result.getRawResult("dollars");
      SlotResult cents = result.getRawResult("cents");
/*
      String dollars = slots.getValue("dollars");
      String cents = slots.getValue("cents");
*/
      StringBuffer b = new StringBuffer();
      if (dollars == null) {
        b.append("0.");
        b.append(cents.value.toString());
      } else {
        b.append(dollars.value.toString());
        if (cents != null) {
          b.append(".");
          b.append(cents.value.toString());
        }
      }
      dollars.value = b.toString();
      return dollars;
    }
  };
/*
  static class DateGrammar extends BuiltInGrammar {
    DateGrammar() {
      super("D_Date", new String[]{"date"});
    }
    public Object getValue(Slots slots, String name) throws EventException {
      return super.getValue(slots, name);
    }
  };
*/
  public static Grammar getGrammarByType(String name) {
```

```
    Grammar gram = (Grammar)builtinGrammars.get(name);
    if (gram == null) {
      synchronized (builtinGrammars) {
        gram = (Grammar)builtinGrammars.get(name);
        if (gram == null) {
          gram = findBuiltinGrammar(name);
        }
        if (gram != null) {
          builtinGrammars.put(name, gram);
        }
      }
    }
    return gram;
  }
  static Grammar findBuiltinGrammar(String type) {
    String name = System.getProperty("jde.grammar." + type + ".name");
    if (name != null) {
      String event = System.getProperty("jde.grammar." + type + ".event");
      if (event != null) {
        return new LinkGrammar(name, event);
      }
      String slot = System.getProperty("jde.grammar." + type + ".slot");
      if (slot == null) {
        slot = type;
      }
      return new BuiltInGrammar(name, new String[]{slot});
    }
    if ("boolean".equals(type)) {
      return new BooleanGrammar();
    } else if ("money".equals(type) ||
          "dollar".equals(type)) {
      return new MoneyGrammar();
    } else {
      LogFactory.getLogFor("vxml").error("no embedded grammar for " + type);
      return null;
    }
/*
    String className = "com.genmagic.invui.vxml.grammar." +
      name + ".Grammar";
    try {
      Class klass = Class.forName(className);
      if (Grammar.class.isAssignableFrom(klass)) {
        return (Grammar)klass.newInstance();
      }
    } catch (Exception ex) {
    }
    return null;
*/
  }
  private static Hashtable builtinGrammars = new Hashtable();
  private static String defaults[][] = {
    {"digit.name", "D_Digit"},
    {"digit.slot", "digit"},
    {"digits.name", "D_Digits"},
    {"digits.slot", "digits"},
    {"number.name", "D_Number"},
```

```
        {"number.slot", "number"},
        {"date.name", "D_Date"},
        {"date.slot", "date"},
        {"time.name", "D_Time"},
        {"time.slot", "time"},
        {"cancel.name", "D_Cancel"},
        {"cancel.event", "cancel"},
        {"help.name", "D_Help"},
        {"help.event", "help"},
        {"phone.name", "D_Phone"},
        {"phone.slot", "phone"},
    };
    static {
        for (int i=0; i<defaults.length; i++) {
            String prop = "jde.grammar."+defaults[i][0];
            if (System.getProperty(prop) == null) {
                System.getProperties().put(prop, defaults[i][1]);
            }
        }
        builtinGrammars.put("none", new NoneGrammar());
        builtinGrammars.put("empty", new NoneGrammar());
    }
}
```

Document ID: 20020204-US-NP

```
package com.genmagic.invui.vxml.dom;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
abstract public class Catch extends VXMLElement {
   public Catch() {
      super("catch");
   }
   public Catch(String name) {
      super(name);
   }
   abstract public String getEventName();
   abstract public int getCount();

   abstract public Expr getCond() throws EventException;
   abstract public boolean match(VXMLInterp ip, String eventName) throws EventException ;
}
```

```
package com.genmagic.invui.vxml.tags;
public class CatchTag extends ExecutableTag {
}
```

Oracle Ex. 2020, p. 20

```
package com.genmagic.invui.vxml.dom;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.VXMLInterp;
/**
 *
 */
abstract public class Choice extends VXMLElement implements DocumentFetchable {
   // TBD
   public Choice() {
      super("choice");
   }
   /**
    * GM extension
    * @deprecated as of Vxml1.0
    */
   abstract public String getName();
   /**
    *
    */
   abstract public String getDtmf();
   /**
    * GM extension
    * @deprecated as of Vxml1.0
    */
   abstract public String getSrc();
   /**
    * GM extension
    * @deprecated as of Vxml1.0
    */
   abstract public String getContentType();
   /**
    *
    */
   abstract public String getNext();
   /**
    *
    */
   abstract public Expr getExpr() throws EventException;
   /**
    * The child content
    */
   abstract public String getContent();
   /**
    * The child content
    */
   abstract public com.genmagic.invui.vxml.interp.Grammar getGrammar(VXMLInterp ip) throws EventException;
}
```

```
package com.genmagic.invui.vxml.dom;
abstract public class Clear extends VXMLElement {
   public Clear() {
      super("clear");
   }
   abstract public String[] getNameList();
   /**
    * @deprecated as of Vxml1.0
    */
   abstract public String[] getName();
}
```

```
package com.genmagic.invui.vxml.tags;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.dom.Clear;
import com.genmagic.invui.vxml.dom.Dialog;
import com.genmagic.invui.vxml.dom.Form;
import com.genmagic.invui.vxml.dom.FormItem;
import com.genmagic.invui.vxml.interp.TagHandler;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLConstants;
import com.genmagic.invui.vxml.interp.VXMLException;
import com.genmagic.invui.vxml.interp.EventException;
public class ClearTag implements TagHandler {
  public void perform(VXMLInterp ip, Element self) throws VXMLException {
    Clear clear = (Clear)self;
    String[] names = clear.getNameList();
    Dialog dialog = ip.getCurrentDialog();
    if (dialog instanceof Form == false) {
      throw new EventException("error.internal",
                    "form not found");
    }
    Form form = ((Form)dialog);
    if (names == null) {
      // names = form.getAllGuardVariables();
      FormItem[] items = form.getFormItems();
      for (int i=0; i<items.length; i++) {
        FormItem item = items[i];
        item.getEventCounter().reset();
        ip.clearVariable(VXMLConstants.DIALOG, item.getName());
      }
    } else {
      ip.clearVariables(VXMLConstants.DIALOG, names);
      for (int i=0; i<names.length; i++) {
        FormItem item = form.getFormItem(names[i]);
        if (item == null) {
          throw new EventException("form item not found for " +
                        names[i]);
        }
        item.getEventCounter().reset();
      }
    }
  }
}
```

```
package com.genmagic.invui.vxml.prompt;
/**
 * CompositePrompt is a abstract interface to allow the composite prompt
 * generator to construct audio/tts/silence in implementation independent
 * mannar.
 *
 * VXML Interpreter may implement this class to generate MTS prompts
 * while JSP component may implement this to generate VoiceXML
 * tags such as <prompt/><audio/><break/>
 *
 */
public interface CompositePrompt {
  /**
   * Append string to be played using TTS
   * @param tts the content to be played by TTS
   */
  public void appendTTS(String tts);
  /**
   * Append dtmf string to be played
   * @param dtmf digits the content to be played
   */
  public void appendDTMF(String dtmf);
  /**
   * Append audio filename to be played. The alternative
   * TTS string which will be played in case where the specified
   * audio file does not exist.
   * @param src the file of audio file
   * @param alt the alternative TTS string
   */
  public void appendAudio(String src, String alt);
  /**
   * Append silence in millisecond.
   * @param time the duration of silence in millisecond
   */
  public void appendSilence(int time);
  /**
   * Append itself to given composite prompt.
   * this will not clear its own content.
   * @param time the duration of silence in millisecond
   * @see clearPrompts
   * @see getCompositePrompt();
   */
  public void appendTo(CompositePrompt set);
  /**
   * Empty the all prompts.
   */
  public void clearPrompts();
  /**
   * Return the clone of composite prompt. The returned instance
   * does not share the sequence of prompts, which means that
   * appending more prompts to the original would not affect
   * the cloned composite prompt.
   * @return the cloned composite prompt.
   */
  public CompositePrompt getCompositePrompt();
  /**
```

```
     * Return the clone of composite prompt. The returned instance
     * does not share the sequence of prompts, which means that
     * appending more prompts to the original would not affect
     * the cloned composite prompt.
     * @param clear clear prompts if this is true.
     * @return the cloned composite prompt.
     */
    public CompositePrompt getCompositePrompt(boolean clear);
    /**
     * Collect tts string from all sub tts prompts in it. The
     * alternative TTS is not included in this string.
     * @return the tts string from all sub tts prompt.
     */
    public String getAllTTS();
    /**
     *
     */
    public boolean getBargein();
    /**
     *
     */
    public void setBargein(boolean b);
}
```

Document Page

```
package com.genmagic.invui.vxml.dialogengine;
import java.util.Enumeration;
import java.util.StringTokenizer;
import java.util.Vector;
import java.net.URL;
import java.net.URLConnection;
import com.genmagic.invui.util.Log;
import com.genmagic.invui.util.LogFactory;
/**
 * This class manages the cookies for a session. Each cookie is created with a
 * Set-Cookie header response from the host server. Cookies have a name with
 * an associated value, path, and domain.
 *
 * @author Paul Hunter
 **/
public class CookieManager {

   static private Vector _topLevelDomains;

   private Vector  _cookies;
   /**
    * Creates an object of this class without any cookies.
    **/
   public CookieManager () {

      _cookies = new Vector(5);

   }
   /**
          * Gets the cookie from the host server's header response. The cookie is
    * defined with the <b>Set-Cookie</b> label.
    *
    * @param url The pointer to the host server.
    * @param conn The connection to the host server based on the URL.
    **/
   public void getCookie (URL url, URLConnection conn) {
      String key;
      int index=1;
      while ( (key=conn.getHeaderFieldKey(index)) != null) {
         if ("set-cookie".equalsIgnoreCase(key)) {
            String cookieString = conn.getHeaderField(index);
            parse(cookieString, url, _cookies);
            System.out.println("receiving cookie = " + cookieString);
         }
         index ++;
      }
      // String cookieString = conn.getHeaderField("Set-Cookie");
/*
      if (cookieString !=  null) {
         parse(cookieString, url, _cookies);
         System.out.println("receiving cookie = " + cookieString);
      }
*/
   }
   /**
    * Send the cookie information being managed by this object to the given
```

```
 * URL. The information contains the name and values for the cookies that
 * apply to the given domain and the default path for the URL connection.
 *
 * @param url The url pointer.
 * @param conn The URL connection to be used to send the cookie
 * information.
 **/
public void sendCookie(URL url, URLConnection conn) {
   String filePath = url.getFile();
   int    index = filePath.lastIndexOf("/");
   if (index <= 0)
      sendCookie(url, conn, "/");
   else
      sendCookie(url,conn, filePath.substring(0, index));

} //* sendCookie
/**
 * Send the cookie information being managed by this object to the given
 * URL. The information contains the name and values for the cookies that
 * apply to the given domain and the given path.
 *
 * @param url The url pointer.
 * @param conn The URL connection to be used to send the cookie
 * information.
 * @param path The path that applies to the cookies to be sent for the
 * given domain.
 **/
public void sendCookie(URL url, URLConnection conn, String path) {
   String domain = url.getHost();
   if (_cookies.size() > 0) {
      StringBuffer buffer = new StringBuffer();
      Enumeration  list = _cookies.elements();

      while (list.hasMoreElements()) {
         Cookie cookie = (Cookie)list.nextElement();
         if (cookie.matches(domain, path)) {
            buffer.append(cookie.getName());
            buffer.append("=");
            buffer.append(cookie.getValue());

         if (list.hasMoreElements())
            buffer.append("; ");
         }
      }
      if (buffer.length() > 0) {
         // System.out.println("sending cookie = '" + buffer.toString() + "'");

         conn.setRequestProperty("Cookie", buffer.toString());
      }
   }

}
/**
 * Returns the cookie in the response header.
 *
 * @param cookieString The <b>Set-Cookie</b> string.
```

```
* @param url The pointer to the host server.
* @param cookies The list of cookies for this manager.
**/
private void parse (String cookieString, URL url, Vector cookies) {

  Cookie        cookie = null;
  String        domain = url.getHost(); //* default
  String        name = null;
  String        nameValue = null;
  String        path = parsePath(url.getFile());
  StringTokenizer st = new StringTokenizer(cookieString, ";");

  /*
   * Parse the Set-Cookie string to find the cookie's attributes.
   */
  while (st.hasMoreTokens()) {
    String tok = st.nextToken().trim();
    int eq = tok.indexOf('=');

    if (eq < 0) {
      name = tok;
    }
    else {

      String key = tok.substring(0, eq).trim();
      String value = tok.substring(eq + 1).trim();

      if ("path".equalsIgnoreCase(key))
        path = value;
      else if ("domain".equalsIgnoreCase(key))
        domain = value;
      else if ("expires".equalsIgnoreCase(key))
        continue;
      else {
        /*
         * If the start of a new cookie is found, create and
         * record the current cookie and start the new one.
         */
        if (name != null) {
          cookie = new Cookie(name, nameValue, path, domain);
          insertCookieEntry(cookies, cookie);
        }
        name = key;
        nameValue = value;
      }
    }
  }
  cookie = new Cookie(name, nameValue, path, domain);
  insertCookieEntry(cookies, cookie);
  // System.out.println(cookies);
}
/**
 * Removes the file from the given path and returns the directory path.
 *
 * @param file The file path.
 * @return A String containing the directory path.
```

```
**/
private String parsePath (String file) {
   int    index = file.lastIndexOf('/');
   String path;

   /**
    * Set the default path to the domain path.
    */
   if (index == -1)
      path = "/";
   else
      path = file.substring(0, index);
   return path;

}

/**
 * Inserts the given cookie entry in the given cookie list.
 *
 * @param entries The entry list for the cookies.
 * @param cookie The cookie to add.
 **/
private void insertCookieEntry (Vector entries, Cookie cookie) {
   int     n = entries.size();
   boolean done = false;
   for (int i = 0; (i < n) && (!done); i++) {
      Cookie existing = (Cookie)entries.elementAt(i);
      if ((existing.getName().equals(cookie.getName())) &&
         (existing.getDomain().equals(cookie.getDomain())) &&
         (existing.getPath().equals(cookie.getPath()))) {
         entries.setElementAt(cookie, i);
         done = true;
      }
   }
   if (!done)
      entries.addElement(cookie);
}

/**
 * This class represents one cookie element in a Set-Cookie/Cookie header
 * line. The name of the cookie and the value for the name are the
 * significant portions of the cookie. The path and domain may be present.
 * If not, their defaults are the root path and the is the host name of
 * the server which generated the cookie response, respectively.
 **/
class Cookie {

   String _domain;
   String _name;
   String _path;
   String _value;
   /**
    * Creates an object of this class. Attributes should either be set to
    * values or to null references.
    *
    * @param name The cookie's name.
```

```
 * @param value The cookie's value.
 * @param path The path on which the cookie is valid.
 * @param domain The domain in which the cookie is valid.
 **/
public Cookie (String name, String value, String path,
               String domain) {

  _name = name;
  _value = value;

  /*
   * Default path is the root path.
   */
  if (path == null)
    _path = "/";
  else
    _path = path;

  _domain = domain;
} //* CookieElement

/**
 * Gets the name of the cookie.
 *
 * @return A String containing the cookie's name.
 **/
public String getName () {

  return _name;

} //* getName

/**
 * Gets the value of the cookie. If the name doesn't have a value, a
 * null reference is returned.
 *
 * @return A String containing the cookie's name value.
 **/
public String getValue () {

  return _value;

} //* getValue

/**
 * Gets the path for which this cookie is valid.
 *
 * @return A String containing the path for the cookie.
 **/
public String getPath () {

  return _path;

} //* getPath

/**
```

```
 * Gets the domain for which this cookie is valid.
 *
 * @return A String containing the domain for the cookie.
 **/
public String getDomain () {

  return _domain;

} //* getDomain

/**
 * Returns true if the cookie is defined for the given path and domain.
 * The cookie's path must be a parent of the given path, and the
 * cookie's domain must match the tail of the given domain.
 *
 * @param domain The domain being checked.
 * @param path The path being checked.
 * @return True if the cookie belongs with the given domain and path.
 **/
public boolean matches (String domain, String path) {
  boolean match = true;
  if ((!path.startsWith(getPath())) ||
    (!domain.endsWith(getDomain())) ||
    (!qualifiedDomain(domain))) {
    match = false;
  }
  return match;

}

/**
 * Returns a string containing the attributes of this object.
 *
 * @return A String containing the cookie's name and attributes.
 **/
public String toString () {

  StringBuffer buffer = new StringBuffer(getName());

  buffer.append("=");
  buffer.append(getValue());
  buffer.append("; domain=");
  buffer.append(getDomain());
  buffer.append("; path=");
  buffer.append(getPath());

  return buffer.toString();

}


/**
 * Returns true if the given domain is a qualified domain for a cookie.
 * The domain must have at least two or three periods. Any domain that
 * fails within one of the seven special top-level domains listed
 * below only require two periods. Any other domain requires at least
```

```
     * three. The seven special top level domains are: <i>com</i>,
     * <i>edu</i>, <i>net</i>, <i>org</i>, <i>gov</i>, <i>mil</i>, and
     * <i>int</i>.
     *
     * @param domain The domain to check.
     **/
    private boolean qualifiedDomain (String domain) {
      return true;
//TODO Need to fix how domains are matched per specification
//      boolean qualified = false;
//      StringTokenizer tokenizer = new StringTokenizer(domain, ".");
//      Vector domainParts = new Vector(3);
//
//      while (tokenizer.hasMoreTokens())
//        domainParts.addElement(tokenizer.nextToken());
//
//      if (((_topLevelDomains.contains(domainParts.lastElement())) &&
//        (domainParts.size() >= 2)) ||
//        (domainParts.size() >= 3))
//        qualified = true;
//
//      return qualified;

    } //* qualifiedDomain */
  } //* class Cookie */

  static {

    _topLevelDomains = new Vector(7);
    _topLevelDomains.addElement("com");
    _topLevelDomains.addElement("org");
    _topLevelDomains.addElement("net");
    _topLevelDomains.addElement("edu");
    _topLevelDomains.addElement("gov");
    _topLevelDomains.addElement("mil");
    _topLevelDomains.addElement("int");

  } //* static */

  public static void main(String a[]) throws Exception {
    URL        url = new URL(a[0]);
    URLConnection con = url.openConnection();
    CookieManager manager = new CookieManager();
    if (a.length == 1) {
      manager.getCookie(url, con);
      manager.sendCookie(url, con, "/");
    }
    else
      if ((a.length == 2) &&
        (a[1].equals("test"))) {
        manager.getCookie(url, con);
        manager.sendCookie(url, con);
/*
        manager.setDebugString("PART_NUMBER=ROCKET_LAUNCHER_0001; path=/");
        manager.getCookie(url, con);
        manager.sendCookie(url, con);
```

```
            manager.setDebugString("PART_NUMBER=ROCKET_LAUNCHER_0002; path=/; SHIPPING=FEDEX;
path=/foo");
            manager.getCookie(url, con);
            manager.sendCookie(url, con);
            manager.setDebugString("PART_NUMBER=ROCKET_LAUNCHER_0001; path=/foo");
            manager.getCookie(url, con);
            manager.sendCookie(url, con);
            manager.setDebugString("PART_NUMBER=RIDING_ROCKET_0023; path=/ammo");
            manager.getCookie(url, con);
            manager.sendCookie(url, con, "/foo");
            manager.sendCookie(url, con, "/ammo");
*/
        } //* if
    } //* main
} //* class CookieManager
```

```
package com.genmagic.invui.vxml.dialogengine;
import java.util.Vector;
import java.util.Enumeration;
import com.genmagic.invui.vxml.prompt.CompositePrompt;
import com.genmagic.invui.vxml.idl.invui.*;
import com.genmagic.invui.vxml.idl.invui.PromptPackage.*;
public class CORBACompositePrompt implements CompositePrompt, Cloneable {
  /***
   * Composite
   ***/
  private Vector  _prompts = new Vector();
  private int     _backupPromptsPos = 0;
  private boolean _bargein = true;
  private int     _storedCount = 1;
  private int     _currentCount = 1;
  private int     _timeout = -1;
  private Vector  _noBargeinPrompts = new Vector();
  private void appendPrompt(Prompt p) {
    if (_bargein) {
      _prompts.addElement(p);
    } else {
      if (_prompts.size()>0) {
        Enumeration e = _prompts.elements();
        while (e.hasMoreElements()) {
          _noBargeinPrompts.addElement(e.nextElement());
        }
        _prompts.setSize(0);
      }
      _noBargeinPrompts.addElement(p);
    }
  }
  protected void appendPrompts(Prompt[] p) {
    for(int i=0; i<p.length; i++) {
      appendPrompt(p[i]);
    }
  }
  public void appendTo(CompositePrompt set) {
    boolean orig = set.getBargein();
    set.setBargein(false);
    appendTo(set, _noBargeinPrompts);
    set.setBargein(true);
    appendTo(set, _prompts);
    set.setBargein(orig);
  }
  static private void appendTo(CompositePrompt set, Vector v) {
    Enumeration e = v.elements();
    while(e.hasMoreElements()) {
      Prompt p = (Prompt)e.nextElement();
      switch (p._discriminator().value()) {
      case PromptType._T_AUDIO:
        set.appendAudio(p.audio().prompt,
                        p.audio().alternative_tts);
        break;
      case PromptType._T_AUDIO_ABS:
        set.appendAudio(p.audio_abs().prompt,
                        p.audio_abs().alternative_tts);
```

```
            break;
         case PromptType._T_TTS:
            set.appendTTS(p.tts());
            break;
         case PromptType._T_SILENCE:
            set.appendSilence((int)p.length());
            break;
         case PromptType._T_DTMF:
            set.appendDTMF(p.dtmf());
         }
      }
   }
   public void clearPrompts() {
      _prompts.setSize(0);
      _noBargeinPrompts.setSize(0);
      _backupPromptsPos = 0;
   }
   public void clearNoBargeinPrompts() {
      _noBargeinPrompts.setSize(0);
   }
   public void appendTTS(String tts) {
      tts = trim(tts);
      if (tts.length()>0) {
         Prompt p = new Prompt();
         p.tts(tts); // + " ");
         appendPrompt(p);
      }
   }
   public void appendDTMF(String dtmf) {
      Prompt p = new Prompt();
      p.dtmf(dtmf);
      appendPrompt(p);
   }
   public void appendAudio(String src, String alt) {
      Prompt p = new Prompt();
      if (alt == null) {
         alt = "";
      }
      if (isAbsPath(src)) {
         p.audio_abs(new audio_abs_s(src, alt));
      } else {
         p.audio(new audio_s(src, alt));
      }
      appendPrompt(p);
   }
   public void appendSilence(int time) {
      Prompt p = new Prompt();
      p.length(time);
      appendPrompt(p);
   }
   public Prompt[] getNoBargeinPrompts() {
      Prompt[] p = new Prompt[_noBargeinPrompts.size()];
      _noBargeinPrompts.copyInto(p);
      return p;
   }
   public Prompt[] getPrompts() {
```

```
    int s = _noBargeinPrompts.size();
    Prompt[] p = new Prompt[s + _prompts.size()];
    if (s>0) {
      _noBargeinPrompts.copyInto(p);
      Enumeration e = _prompts.elements();
      while(e.hasMoreElements()) {
        p[s++] = (Prompt)e.nextElement();
      }
    } else {
      _prompts.copyInto(p);
    }
    return p;
  }
  public CompositePrompt getCompositePrompt() {
    return getCompositePrompt(false);
  }
  public CompositePrompt getCompositePrompt(boolean clear) {
    if (clear == false) {
      // do not clear
      CompositePrompt impl = new CORBACompositePrompt();
      appendTo(impl);
      return impl;
    } else {
      // clear
      CompositePrompt impl = new CORBACompositePrompt();
      appendTo(impl);
      clearPrompts();
      return impl;
    }
  }
  public void backupPrompts() {
    _backupPromptsPos = _prompts.size();
  }
  public void setCount(int count) {
    //System.out.println("set count = " + count);
    _currentCount = count;
    _storedCount = 1;
  }
  public int getCount() {
    return _currentCount;
  }
  public boolean checkCount(int c) {
    if (_currentCount <0) {
      // count is meaning less in executalbe
      return true;
    }
    // System.out.println("check count = " + c + ", storedCount = " +
    // storedCount + ", currentCount =" + currentCount);
    if (_storedCount == c) {
      // System.out.println("ok");
      // ok
      return true;
    } else if (c > _currentCount || // if it's more than current count
          _storedCount > c) {  // if it's less than latest count
      //System.out.println("no");
      return false;
```

```
    } else { // if it's new count..
      _prompts.setSize(_backupPromptsPos); // keep the previous prompts
      _storedCount = c;
      _timeout = -1;
      //System.out.println("change");
      return true;
    }
  }
  public void setTimeout(int to) {
    _timeout = to;
  }
  public int getTimeout() {
    return _timeout;
  }
  public void clearTimeout() {
    setTimeout(-1);
  }
  public void setBargein(boolean b) {
    _bargein = b;
  }
  public boolean getBargein() {
    return _bargein;
  }
  public String getAllTTS() {
    StringBuffer buffer = new StringBuffer();
    getAllTTS(_noBargeinPrompts, buffer);
    getAllTTS(_prompts, buffer);
    return buffer.toString();
  }
  static private void getAllTTS(Vector v, StringBuffer buffer) {
    Enumeration e = v.elements();
    while(e.hasMoreElements()) {
      Prompt p = (Prompt)e.nextElement();
      if (p._discriminator() == PromptType.T_TTS) {
        buffer.append(p.tts());
      }
    }
  }
  static String trim(String str) {
    if (str == null) {
      return null;
    }
    int len = str.length();
    int i = 0;
    for (; i<len; i++) {
      char c = str.charAt(i);
      if (Character.isWhitespace(c) ||
        c == '\n' || c == '\r') {
        continue;
      } else {
        break;
      }
    }
    str = str.substring(i);
    for (i = str.length()-1; i>=0; i--) {
      char c = str.charAt(i);
```

```
            if (Character.isWhitespace(c) ||
                c == '\n' || c == '\r') {
                continue;
            } else {
                break;
            }
        }
        return str.substring(0, i+1);
    }
    static private boolean isAbsPath(String str) {
        if (str.startsWith("\\") ||
            str.startsWith("/") ||
            (str.length()>1 &&  str.charAt(1) == ':')) {
            return true;
        }
        return false;
    }
}
```

Document Page 3 of 3

```
package com.genmagic.invui.vxml.prompt;
import java.io.*;
import java.util.Hashtable;
import java.util.Properties;
public class CustomPromptGenerator implements PromptGenerator {
   Hashtable map = new Hashtable();
   protected CustomPromptGenerator(String[][] m) {
      for (int i=0; i<m.length; i++) {
         if (m[i].length == 2) {
            map.put(m[i][0], m[i][1]);
         }
      }
   }
   protected CustomPromptGenerator(File f) throws IOException {
      this(new FileInputStream(f));
   }
   protected CustomPromptGenerator(InputStream in) throws IOException {
      Properties p = new Properties();
      p.load(in);
      map = p;
   }
   public void appendPrompt(CompositePrompt set, String content) {
      String wav = (String)map.get(content.trim());
      if (wav != null) {
         set.appendAudio(wav, content);
      } else {
         set.appendTTS(content);
      }
   }
}
```

```
package com.genmagic.invui.vxml.dialogengine;;
public interface DEConstants {
   public final static long OUTDIAL_TIMEOUT = 30000;
   public final static int SUCCESS = 0;
   public final static int IN_USE = 45400;
   public final static int IDLE = IN_USE + 1;                    // 45401
   public final static int NO_ANSWER = IDLE + 1;      // 45402
   public final static int BUSY = NO_ANSWER + 1;     // 45403
   public final static int ERROR = BUSY + 1;            // 45404
   public final static int MISRECOGNITION = ERROR + 1;    // 45405
   public final static int NO_NUMBER = MISRECOGNITION + 1; // 45406
   public final static int QUEUED = NO_NUMBER + 1;       // 45407
   public final static int NO_LINE = QUEUED + 1;         // 45408
   public final static int TIMEOUT = NO_LINE + 1;        // 45409
/*
   public final static int CALL_CONNECTED = 0;
   public final static int CALL_BUSY     = 1;
   public final static int CALL_TIMEOUT   = 2;
   public final static int CALL_NO_ANSWER = 3;
   public final static int CALL_ERROR     = 4;

   public final static String CALL_STATUS[] = {
      "CONNECTED",
      "BUSY",
      "TIMEOUT",
      "NO_ANSWER",
      "ERROR",
   };
*/
/*
   public final static int BAD_TYPE = 0;
   public final static int HOME = 1;
   public final static int WORK = 2;
   public final static int MOBILE = 3;
   public final static int CURRENT = 4;
   public final static int BACKUP = 5;
   public static final String PHONE_TYPE_STRING[] = {
      "BAD_TYPE",
      "HOME",
      "WORK",
      "MOBILE",
      "CURRENT",
      "BACKUP",
   };
*/
}
```

```
package com.genmagic.invui.vxml.dom;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
abstract public class Dialog extends VXMLElement {
   public Dialog(String name) {
      super(name);
   }
   /**
    * Gets ID
    */
   abstract public String getID();
   /**
    * Gets the scope of form grammars
    */
   abstract public int getScope();
   /**
    * @deprecated as of 1.0
    */
   abstract public String getCost();
   /**
    * @deprecated as of 1.0
    */
   abstract public boolean getBargein();
   /**
    * @deprecated as of 1.0
    */
   abstract public int getTimeout();
   /**
    * Grammar
    */
   abstract public com.genmagic.invui.vxml.interp.Grammar[] getDocumentGrammars(VXMLInterp ip) throws
EventException;
   /**
    * Gets event handler (catch)
    */
   abstract public Catch getEventHandler(VXMLInterp ip, Exception ev, int count) throws EventException;
}
```

```
package com.genmagic.invui.vxml.dialogengine;
import java.net.*;
import java.util.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.Messaging.*;
import com.genmagic.invui.util.Log;
import com.genmagic.invui.util.LogFactory;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.SpeechControl;
import com.genmagic.invui.vxml.interp.VXMLConstants;
import com.genmagic.invui.vxml.interp.VXMLException;
import com.genmagic.invui.vxml.interp.XMLLoader;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.AppContext;
import com.genmagic.invui.vxml.interp.ResourceManager;
import com.genmagic.invui.vxml.interp.Util;
import com.genmagic.invui.vxml.idl.invui.*;
public class DialogEngine extends DialogEnginePOA implements DEConstants, VXMLConstants
{
   private ORB orb = null;
   private boolean emulator = false;
   private String inURL = null;
   private boolean outgoing = false;
   private boolean logging = true;
   private int max;
   private Hashtable sessions = new Hashtable();
   private VUIClientImpl vuiclient;
   private ResourceManager resourceManager = new ResourceManagerFileImpl();
   static final Log log = LogFactory.getLogFor("jde");
/*
   public DialogEngine(String in, String slotdefs[][], String group) throws Exception {
      this(in, new Connection(slotdefs, max), p);
      emulator = true;
   }
*/
   public DialogEngine(ORB o, String jde_group, String mg) throws Exception {
      orb = o;
/*
      com.inprise.vbroker.CORBA.BindOptions   bindOptions = new com.inprise.vbroker.CORBA.BindOptions();
      bindOptions.defer_bind     = false;
      bindOptions.enable_rebind  = true;
      ((com.inprise.vbroker.orb.ORB) orb).default_bind_options(bindOptions);
*/
      String mts_group = mg;
      if (jde_group == null ||
         jde_group.trim().length() == 0) {
         jde_group = mts_group;
      }
      vuiclient = new VUIClientImpl(this, orb, mts_group);
      // Initialize the POA.
      POA rootPOA = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
      // Create policies for our persistent POA
      org.omg.CORBA.Policy[] policies = {
         rootPOA.create_lifespan_policy(LifespanPolicyValue.PERSISTENT)
      };
```

```
  // ACTIVATE VUICLIENT ----------------------------
  // Create vcPOA with the right policies
  POA vcPOA = rootPOA.create_POA("VUIClient_" + jde_group + "_poa",
                        rootPOA.the_POAManager(),
                        policies );
  // Decide on the ID for the servant
  // Activate the servant with the ID on vcPOA
  vcPOA.activate_object_with_id(("VUIClient." + jde_group).getBytes(),
                        vuiclient);
  // ACTIVATE JDE ----------------------------
  // Create jdePOA with the right policies
  POA jdePOA = rootPOA.create_POA("JDE_" + jde_group + "_poa", rootPOA.the_POAManager(),
                        policies );
  // Decide on the ID for the servant
  // Activate the servant with the ID on myPOA
  jdePOA.activate_object_with_id(("JDE." + jde_group).getBytes(),
                        this);
  // Activate the POA manager
  rootPOA.the_POAManager().activate();
  org.omg.CORBA.Object vobj = vcPOA.servant_to_reference(vuiclient);
  org.omg.CORBA.Object dobj = jdePOA.servant_to_reference(this);
  String v_ior = orb.object_to_string(vobj);
  String d_ior = orb.object_to_string(dobj);
  log.info("starting VUIClient REF =" + vobj);
  log.info("       VUIClient ID  =" + v_ior);
  log.info("starting DialogEngine REF =" + dobj);
  log.info("       DialogEngine ID  =" + d_ior);
  System.out.println("VUICLIENT = " + vobj +
            "\n     ID = " + orb.object_to_string(vobj) +
            "\nJDE     = " + dobj +
            "\n     ID = " + orb.object_to_string(dobj) +
            " are ready.");
}
public Call placeCall(String number, String sub, byte[] uui) throws Exception {
  return vuiclient.placeCall(number, sub, uui, 30000);
}
public VXMLInterp outgoingCall(String url, String number,
                        String caller,
                        byte[] uui,
                        int timeout) throws Exception {
  Call call = vuiclient.placeCall(number, caller, uui, timeout);
  XMLLoader loader = new XMLLoaderImpl(url);
  // loader.addParameter("direction", "out");
  AppContext app = new AppContext(loader, resourceManager);
  VXMLInterp ip = new VXMLInterpCORBAImpl(app, this, call);
  SpeechControl speech = ip.getSpeechControl();
  if (logging) {
    speech.enableLog();
  }
  // ip.setCallInfo(null, null, null);
  addSession(call, ip);
  return ip;
}
public void setAcceptIncomingCall(String in) {
  inURL = in;
}
```

```
  public void setAcceptOutgoingCall() {
    outgoing = true;
  }
  public void setMaxNumberOfCalls(int m) {
    max = m;
  }
  public void setMTSLog(boolean b) {
    logging = b;
  }
  public boolean handleCall(Call c, String dnis, String ani, byte[] UUI) {
    log.info("[" + c.id + "] incoming call : dnis=" +
          dnis +", ani=" + ani + ", uui(base64encoded)=" + Util.encodeBASE64(UUI));
    // if (max <= current || inURL == null) {
    if (inURL == null) {
      System.out.print("rejectingCall");
      return false;
    }
    XMLLoader loader = null;
    try {
      loader = new XMLLoaderImpl(inURL);
    } catch (org.xml.sax.SAXException ex) {
      ex.printStackTrace();
      return false;
    } catch (java.io.IOException ex) {
      ex.printStackTrace();
      return false;
    }
    loader.addParameter("telephone.dnis", dnis);
    loader.addParameter("telephone.ani", ani);
    loader.addParameter("telephone.uui", UUI);
    // loader.addParameter("direction", "in");
    AppContext app = new AppContext(loader, resourceManager);
    VXMLInterp interp = new VXMLInterpCORBAImpl(app, this, c);
    // the initial scope is session scope
    SpeechControl speech = interp.getSpeechControl();
    if (logging) {
      speech.enableLog();
    }
    try {
      interp.setCallInfo(ani, dnis, UUI);
      // speech.setInfo(dnis, ani);
    } catch (Exception ex) {
      return false;
    }
    addSession(c, interp);
    new Thread(interp).start();
    return true;
  }
  public void placeCall(String number, String caller, String url, boolean sync) throws VXMLRuntimeError,
DocumentNotFound, MalformedURL {
    try {
      byte[] uui = new byte[0];
      VXMLInterp ip = outgoingCall(url, number, caller, uui, 30000);
/*
      // for JDE1.0 (VXML0.9)
      ip.createVariable("dnis");
```

```
            ip.createVariable("ani");

            // for VXML1.0
            ip.createVariable("telephone.dnis");
            ip.createVariable("telephone.ani");
            // ip.createVariable("telephone.direction", "out");
*/
            if (sync) {
               ip.run();
            } else {
               new Thread(ip).start();
            }
/*
         } catch (NoMoreChannel ex) {
            throw ex;
         } catch (UserBusy ex) {
            throw ex;
         } catch (NetworkBusy ex) {
            throw ex;
         } catch (NetworkError ex) {
            throw ex;
         } catch (InvalidNumber ex) {
            throw ex;
         } catch (TelephonyError ex) {
            throw ex;
*/
         } catch (VXMLException ex) {
            throw new VXMLRuntimeError();
         } catch (MalformedURLException ex) {
            throw new MalformedURL();
         } catch (java.io.IOException ex) {
            throw new DocumentNotFound();
         } catch (SystemException ex) {
            ex.printStackTrace();
            throw ex;
         } catch (Exception ex) {
            ex.printStackTrace();
         }
      }
      public void addSession(Call c, VXMLInterp h) {
         String id = orb.object_to_string(c.session);
         sessions.put(id, h);
      }
      VXMLInterpCORBAImpl getSession(Call c) {
         String id = orb.object_to_string(c.session);
         return (VXMLInterpCORBAImpl)sessions.get(id);
      }
      VXMLInterpCORBAImpl removeSession(Call c) {
         String id = orb.object_to_string(c.session);
         return (VXMLInterpCORBAImpl)sessions.remove(id);
      }
   }
```

```
package com.genmagic.invui.vxml.prompt.generator.digits;
import com.genmagic.invui.vxml.prompt.InvalidFormatException;
import com.genmagic.invui.vxml.prompt.CompositePrompt;
public class DigitsPrompt {
   public static void appendDigitsPrompt(CompositePrompt set, String d, String q) {
      set.appendAudio("dig_" + d + "_" + q + ".wav",
                  d);
   }
}
```

Oracle - Exhibit 2003

```
package com.genmagic.invui.vxml.dom;
/*
 * @since Vxml1.0
 */
public interface DocumentFetchable extends Fetchable {

   /**
    * URI of  the audio clip to play while fetching.
    */
   public String getFetchAudio();
}
```

```
package com.genmagic.invui.vxml.domimpl;
import com.genmagic.invui.vxml.dom.VXMLElement;
class DocumentFetchInfo extends FetchInfo {
   String fetchAudio;
   DocumentFetchInfo(VXMLElement e) {
      super(e);
      fetchAudio = e.getAttribute2("fetchaudio");
   }
}
```

Document ID: 1

```
package com.genmagic.invui.vxml.dom;
abstract public class Dtmf extends Grammar implements Fetchable {
   public Dtmf() {
      super("dtmf");
   }
   abstract public String getSrc();
   abstract public int getScope();
   abstract public String getType();
   abstract public com.genmagic.invui.vxml.interp.Grammar getGrammar();
}
```

Document ID: 20020826.1

```
package com.genmagic.invui.vxml.interp;
/*
import com.genmagic.mts.Session;
import com.genmagic.mts.MTSException;
import com.genmagic.mts.MTSConstants;
*/
import java.util.Vector;
import java.util.Hashtable;
abstract public class DynamicGrammar extends Grammar { // implements MTSConstants {
   abstract public String getKey();
   public String getUniqID() {
      return getKey();
   }
   abstract public String getActiveGrammarNames();
   static Hashtable _keyCache = new Hashtable();
   /**
    * Utitity function to generate key from string.
    */
   static protected String genKey(String str) {
      String key = (String)_keyCache.get(str);
      if (key == null) {
         synchronized(DynamicGrammar.class) {
            key = (String)_keyCache.get(str);
            if (key != null) {
               return key;
            }
            key = Util.genKey(str);
            _keyCache.put(str, key);
         }
      }
      return key;
   }
   /**
    * FOR CMS API. Nuance Specific
    */
   protected static final String DYNAMIC_TOP = ".DynamicTop";
/*
   protected static final String DYNAMIC_ROOT = "DynamicRoot";
   protected static final String[] dynamicGrammarNames= {
      "Dynamic1",
      "Dynamic2",
      "Dynamic3",
      "Dynamic4",
      "Dynamic5",
      "Dynamic6",
      "Dynamic7",
      "Dynamic8",
      "Dynamic9",
      "Dynamic10",
      "Dynamic11",
      "Dynamic12",
      "Dynamic13",
      "Dynamic14",
      "Dynamic15",
      "Dynamic16",
      "Dynamic17",
```

```
    "Dynamic18",
    "Dynamic19",
    "Dynamic20",
    "Dynamic21",
    "Dynamic22",
    "Dynamic23",
    "Dynamic24",
    "Dynamic25",
    "Dynamic26",
  };
  protected static final String[] dynamicGrammarRefs= {
    "DynaRef1",
    "DynaRef2",
    "DynaRef3",
    "DynaRef4",
    "DynaRef5",
    "DynaRef6",
    "DynaRef7",
    "DynaRef8",
    "DynaRef9",
    "DynaRef10",
    "DynaRef11",
    "DynaRef12",
    "DynaRef13",
    "DynaRef14",
    "DynaRef15",
    "DynaRef16",
    "DynaRef17",
    "DynaRef18",
    "DynaRef19",
    "DynaRef20",
    "DynaRef21",
    "DynaRef22",
    "DynaRef23",
    "DynaRef24",
    "DynaRef25",
    "DynaRef26",
  };
  abstract public void install(Session ses) throws MTSException;
  abstract public void insertGrammar(Session ses) throws MTSException;
  public void startRecognition(Session session,
                               int rec_timeout,
                               int toomuch_timeout,
                               int endspeech_timeout,
                               int max_digit) throws com.genmagic.mts.MTSException
  {
    install(session);
    // DynamicGrammar.installGrammar(session, this);
    insertGrammar(session);
    int stat = session.waitDynamicGrammarInsertion();
    if (stat != DG_Success) {
      throw new MTSException("Fail to insert dynamic grammar :" +
                             DG_STATUS_STRING[stat]);
    }
    session.recognition(getName(),
                        rec_timeout,
```

```
                toomuch_timeout,
                endspeech_timeout,
                max_digit,
                getActiveGrammarNames());
        }
    */
    }
```

```
package com.genmagic.invui.vxml.dom;
abstract public class Enumerate extends VXMLElement {
   // TBD
   public Enumerate() {
      super("enumerate");
   }
   abstract public String getStaticContent();
   abstract public void setStaticContent(String content);
}
```

Document Ref: 5PP0-2

```
package com.genmagic.invui.vxml.tags;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.dom.Choice;
import com.genmagic.invui.vxml.dom.Enumerate;
import com.genmagic.invui.vxml.dom.Form;
import com.genmagic.invui.vxml.dom.Menu;
import com.genmagic.invui.vxml.dom.Dialog;
import com.genmagic.invui.vxml.interp.*;
public class EnumerateTag implements TagHandler, VXMLConstants {

   public void perform(VXMLInterp ip, Element self) throws VXMLException {
      Enumerate enum = (Enumerate)self;
      // will be VXMLInterp.getCurrentMenu....?
      Dialog d = ip.getCurrentDialog();
      if (d instanceof Menu) {
      } else if (d instanceof Form) {
         throw new EventException("error.not_supported.enumerate",
                        "enumerate for field is not supported yet");
      } else {
         throw new EventException("error.semantic",
                        "menu/field not found for enumeration");
      }
      Menu menu = (Menu)d;
      Choice[] choices = menu.getChoices();
      if (enum.hasChildNodes()) {
         ip.createVariable("_prompt");
         ip.createVariable("_dtmf");
         for (int i=0; i<choices.length; i++) {
            Choice c = choices[i];
            ip.setVariable("_prompt", c.getContent());
            String dtmf = c.getDtmf();
            if (dtmf == null) {
               dtmf = String.valueOf(i+1);
            }
            ip.setVariable("_dtmf", c.getContent());
            ip.evalChildren(enum);
         }
         ip.removeVariable("_prompt");
         ip.removeVariable("_dtmf");
      } else {
         String content = enum.getStaticContent();
         if (content == null) {
            StringBuffer buf = new StringBuffer();
            int last = choices.length -1;
            for (int i=0; i<choices.length; i++) {
               if (i == last) {
                  buf.append(". or ");
               } else {
                  buf.append(". ");
               }
               buf.append(choices[i].getContent());
            }
            content = buf.toString();
            enum.setStaticContent(content);
         }
```

```
            ip.appendTTS(content);
        }
    }
/*
    static public Menu getCurrentMenu(Node n) throws EventException {
        for (;
            n != null && (n instanceof Menu) == false;
            n = n.getParentNode());
        return (Menu)n;
    }
    static public String toString(Prompt[] p) {
        StringBuffer buffer = new StringBuffer();
        for (int i=0; i<p.length; i++) {
            if (p[i].type == Prompt.TTS) {
                buffer.append(p[i].getTTS());
            }
        }
        return buffer.toString();
    }
*/
}
```

```
package com.genmagic.invui.vxml.script.expr;
import java.util.Hashtable;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.script.Frame;
public class Environment implements ExprConstants, Frame {
  Hashtable bindings = new Hashtable();
  Environment parent = null;
  public Environment(Environment p) {
    parent = p;
  }
  public Environment getParent() {
    return parent;
  }
  public Object getVariable(String name) {
    return bindings.get(name);
  }
  public void setVariable(String name, Object value) throws EventException {
    Object prev = bindings.put(name, value);
    if (prev == null) {
      bindings.remove(name);
      throw new EventException("error.semantic",
                  "variable not found " + name);
    }
  }
  public void createVariable(String name, Object value) {
    bindings.put(name, value);
  }
  public void clearVariable(String name) throws EventException {
    if (bindings.put(name, UNDEFINED_OBJ) == null) {
      bindings.remove(name);
      throw new EventException("error.semantic",
                  "variable not found " + name);
    }
  }
  public void removeVariable(String name) throws EventException {
    if (bindings.remove(name) == null) {
      throw new EventException("error.semantic",
                  "variable not found " + name);
    }
  }
  public boolean hasVariable(String name) {
    return bindings.containsKey(name);
  }
  public java.util.Enumeration getVariableNames() {
    return bindings.keys();
  }
  public boolean hasDefinedVariable(String name) {
    Object obj = bindings.get(name);
    return obj != null && obj != UNDEFINED_OBJ;
  }
/*
  public void clearBindings() {
    bindings.clear();
  }
*/
  public void removeBindings() {
```

```
        bindings.clear();
    }
}
```

Document ID: 0.7.1.20

```
package com.genmagic.invui.vxml.interp;
import java.util.Hashtable;
final public class EventCounter {
   Hashtable table = null;
   int promptCount = 0;
   public int getCount(String eventName) {
      if (table == null) {
         table = new Hashtable();
      }
      Integer i = (Integer)table.get(eventName);
      if (i == null) {
         table.put(eventName, new Integer(1));
         return 1;
      } else {
         int c = i.intValue() + 1;
         table.put(eventName, new Integer(c));
         return c;
      }
   }
   public int getPromptCount() {
      return promptCount;
   }
   public void incrementPromptCounter() {
      ++ promptCount;
   }
   public void reset() {
      if (table != null) {
         table.clear();
      }
      promptCount = 1;
   }
}
```

```
package com.genmagic.invui.vxml.interp;
public class EventException extends VXMLException {
   String eventName;
   Object arg = null;
   public EventException(String n) {
      eventName = n;
   }
   public EventException(String n, String msg) {
      super(msg);
      eventName = n;
   }
   public EventException(String n, String msg, Throwable t) {
      super(msg, t);
      eventName = n;
   }
   public EventException(String n, Object a) {
      eventName = n;
      arg = a;
   }
   public String getMessage() {
      return eventName + " : " + super.getMessage();
   }
   public String getEventName() {
      return eventName;
   }
   public Object getArg() {
      return arg;
   }
}
```

Document Page

```
package com.genmagic.invui.vxml.interp;
import java.util.Enumeration;
import java.util.Vector;
import java.util.Hashtable;
import org.w3c.dom.Node;
import org.w3c.dom.Attr;
import com.genmagic.invui.vxml.dom.VXMLDocument;
import com.genmagic.invui.vxml.dom.Catch;
/**
 */
public class EventHandlerSet {
   Vector[] event_list = null;
   Node node = null;
   public EventHandlerSet(Node n) {
      node = n;
      Vector v = new Vector();
      collectEventHandlers(v, node);
      v.trimToSize();
      event_list = new Vector[v.size()];
      v.copyInto(event_list);
   }
   /**
    * This is slow, any other algorithm?
    */
   public Catch getEventHandler(VXMLInterp ip, Exception ex, int count) throws EventException {
      String eventName = null;
      if (ex instanceof RuntimeEventException) {
         eventName = ((RuntimeEventException)ex).getEventName();
      } else {
         eventName = ((EventException)ex).getEventName();
      }
      // String eventName = ev.getEventName();
      for (int i = Math.min(count, event_list.length) - 1; i>=0; i--) {
         if (event_list[i] != null) {
            Vector v = event_list[i];
            Enumeration e = v.elements();
            while (e.hasMoreElements()) {
               Catch c = (Catch)e.nextElement();
               if (c.match(ip, eventName)) {
                  return c;
               }
            }
         }
      }
      // REMIND: wild card support
      Catch c = (Catch)defaultEventHandlers.get(eventName);
      if (c == null) {
         if (ex instanceof RuntimeEventException) {
            ip.unhandledEvent((RuntimeEventException)ex);
         } else {
            ip.unhandledEvent((EventException)ex);
         }
      }
      return c;
   }
   static final int INCREMENT = 2;
```

```
public static void addEventHandler(Vector column, Catch c) {
   int count = c.getCount() - 1;
   Vector row = null;
   if (column.size() > count) {
      row = (Vector)column.elementAt(count);
   } else {
      column.setSize(count + INCREMENT);
   }
   if (row == null) {
      row = new Vector();
      column.setElementAt(row, count);
   }
   row.addElement(c);
}
public static void collectEventHandlers(Vector column, Node node) {
   for (Node child = node.getFirstChild();
        child != null;
        child = child.getNextSibling()) {
      if (child instanceof Catch) {
         addEventHandler(column, (Catch)child);
      }
   }
   node = node.getParentNode();
   if (node != null) {
      // try parent node for now...
      collectEventHandlers(column, node);
   }
}
private static Hashtable defaultEventHandlers = new Hashtable();
static {
   VXMLDocument _template = new VXMLDocumentImpl(null);
   // VXMLDocument _template = new TestDocument(null);
   Node root = _template.createElement("vxml");
   _template.insertBefore(root, null);
   Attr attr = _template.createAttribute("version");
   attr.setValue("1.0");
   root.getAttributes().setNamedItem(attr);
   root.insertBefore(_template.createComment("default handlers"), null);
   root.insertBefore(_template.createTextNode("\n"), null);
   // Need to support 1.0 and 0.9
   // help
   Catch c = (Catch)_template.createElement("help");
   c.insertBefore(_template.createTextNode("No help is available."), null);
   c.insertBefore(_template.createElement("reprompt"), null);
   defaultEventHandlers.put("help", c);
   root.insertBefore(c, null);
   root.insertBefore(_template.createTextNode("\n"), null);
   // cancel
   c = (Catch)_template.createElement("cancel");
   c.insertBefore(_template.createTextNode("Cancel is not supported here."), null);
   c.insertBefore(_template.createElement("reprompt"), null);
   defaultEventHandlers.put("cancel", c);

   root.insertBefore(c, null);
   root.insertBefore(_template.createTextNode("\n"), null);
   // nomatch
```

```
        c = (Catch)_template.createElement("nomatch");
        c.insertBefore(_template.createTextNode("Sorry. I did not understand what you said."),
                null);
        c.insertBefore(_template.createElement("reprompt"), null);
        defaultEventHandlers.put("nomatch", c);
        root.insertBefore(c, null);
        root.insertBefore(_template.createTextNode("\n"), null);
        // noinput
        c = (Catch)_template.createElement("noinput");
        c.insertBefore(_template.createTextNode("Please say something."), null);
        c.insertBefore(_template.createElement("reprompt"), null);
        defaultEventHandlers.put("noinput", c);
        root.insertBefore(c, null);
        root.insertBefore(_template.createTextNode("\n"), null);
    }
}
```

```
package com.genmagic.invui.vxml.tags;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.interp.TagHandler;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLException;
public class ExecutableTag implements TagHandler {
   public void perform(VXMLInterp ip, Element self) throws VXMLException {
      // count is meaningless in executable content
      // See VoiceXML1.0 specification 19.5 page 72
      ip.getSpeechControl().setCount(-1);
      try {
         ip.beginAnonymousScope();
         ip.evalChildren(self);
      } finally {
         ip.endAnonymousScope();
      }
   }
}
```

```
package com.genmagic.invui.vxml.dom;
import com.genmagic.invui.vxml.interp.EventException;
abstract public class Exit extends VXMLElement {
  public Exit() {
    super("exit");
  }
  /**
   * Get expression.
   */
  abstract public Expr getExpr() throws EventException;
  /**
   * @since Vxml1.0
   */
  abstract public String[] getNameList();
}
```

```
package com.genmagic.invui.vxml.tags;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.dom.Exit;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.interp.TagHandler;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
public class ExitTag implements TagHandler {
   public void perform(VXMLInterp ip, Element self) throws EventException {
      Exit exit = (Exit)self;
      Object obj = null;
      Expr expr = exit.getExpr();
      if (expr != null) {
         obj = ip.evalExpression(expr);
      }
      throw new EventException("exit", obj);
   }
}
```

Document Page Blank

```
package com.genmagic.invui.vxml.dom;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.VXMLInterp;
public interface Expr {
   public String getExpression();
   public java.lang.Object getExpressionObject();
   public void setExpressionObject(java.lang.Object obj);
/*
   public boolean evaluateBoolean(VXMLInterp ip) throws EventException;
   public java.lang.Object evaluate(VXMLInterp ip) throws EventException;
*/
};
```

```
package com.genmagic.invui.vxml.script.expr;
/**
 * This class defines types of a token in an expression.
 */
public interface ExprConstants {
   static class Undefined {
      public String toString() {
         return "undefined";
      }
   }
   static final Object UNDEFINED_OBJ = new Undefined();
}
```

Document ID: 1 Page: 1 of 1

```
package com.genmagic.invui.vxml.script.expr;
import java.util.*;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.lex.*;
/**
 * This class evaluates expressions
 */
final public class Expression implements ExprConstants, LexConstants {
   private ExprToken token = null;
   private Enumeration tokens = null;
   private VXMLInterp m_interp;
   private String source;
/*
   public Expression(VXMLInterp ip, String src) throws EventException {
      this(ip, getEnum(src), src);
   }
   private static Hashtable h = new Hashtable();
   static private Enumeration getEnum(String src) throws EventException {
      Object o = h.get(src);
      if (o == null) {
         o = new Lex(src).tokenize();
         h.put(src, o);
      }
      return ((Vector)o).elements();
   }
*/
   /**
    * @param expr, the epxression we are parsing.
    * @param source
    * @exception ExpressionException usually indicates a syntax error in the expression
    *            also a tag we invoked may throw an exception.
    */
   public Expression(VXMLInterp ip, Enumeration tks, String src) {
      m_interp = ip;
      tokens = tks;
      source = src;
   }
   private void next() throws EventException {
      if (tokens.hasMoreElements()) {
         token = (ExprToken)tokens.nextElement();
         token = token.evaluate(m_interp);
      }
   }
   /** This method evaluates the expression
    * @param input A vector of tokens we will evaluate.
    * @exception If the expression is invalid (bad syntax, etc)
    */
   public Object evaluate() throws EventException {
      next();
      Object value = exprTop();
      if (token.ttype != EOE) {
         throw new ExpressionException("Unexpected " +
                                       LexConstants.TOKEN_TYPE[token.ttype] +
                                       " token = " + token +
                                       " src = " + source);
```

```
    }
    return value;
  }
  /**
   * Evaluate the top most expressoin
   * Node:Chice is right associative
   *
   * exprTop : exprOr ? exprTop : exprTop
   *
   * @exception ExpressionException usually indicates a syntax error in
   *            the expression also a tag we invoked may throw an exception.
   */
  public Object exprTop() throws EventException {
    Object result = exprOr();
    // || has the top priority
    while(token.ttype == CHOICE1) {
      next();
      Object left = exprTop(); // right associative
      if (token.ttype != CHOICE2) {
        throw new ExpressionException(" ':' expected");
      }
      next();
      Object right = exprTop();// right associative
      Boolean b = toBoolean(result);
      if ( b.booleanValue() ) {
        result = left;
      } else {
        result = right;
      }
    }
    return result;
  }
  /**
   * Evaluate or expression
   *  exprOr : exprAnd (|| exprAnd)*
   * @exception ExpressionException usually indicates a syntax error in
   *            the expression also a tag we invoked may throw an exception.
   */
  public Object exprOr() throws EventException {
    Object result = exprAnd();
    while(token.ttype == LOGOR) {
      next();
      Boolean left = toBoolean(result);
      Boolean right = toBoolean(exprAnd());
      result = new Boolean(left.booleanValue() || right.booleanValue());
    }
    return result;
  }
  /**
   * Evaluate the and expression
   * exprAnd : exprComp (&& exprComp)*
   */
  public Object exprAnd() throws EventException {
    Object result = exprEq();
    while(token.ttype == LOGAND) {
      int op = token.ttype;
```

```
      next();
      Boolean left = toBoolean(result);
      Boolean right = toBoolean(exprEq());
      //System.out.println("and");
      result = new Boolean(left.booleanValue() && right.booleanValue());
    }
    return result;
  }
  /**
   * exprEq : expr ( == expr)*
   * (and <= >= != =)
   */
  public Object exprEq() throws EventException {
    Object result = exprRel();
    while (token.ttype == EQOP || token.ttype == ASSIGN) {
      char type = token.ctoken;
      next();
      Object left = result;
      Object right = exprRel();
      boolean bresult = false;
      int t = mathType(left, right);
      //System.out.println("eq or comp " + type);
      switch (type) {
      case '!': // with type conversion
        if (t == BOOLEAN) {
          bresult = !left.equals(right);
        } else if (t == STRING) {
          bresult = !left.toString().equals(right.toString());
        } else if (t == INTEGER) {
          bresult = ((Number)left).longValue() != ((Number)right).longValue();
        } else if (t == FLOAT) {
          bresult = ((Number)left).doubleValue() != ((Number)right).doubleValue();
        } else {
          bresult = ! left.equals(right);
          // throw new ExpressionException("Unexpected operand types for '!='");
        }
        break;
      case '?': // strict
        bresult = ! left.equals(right);
        break;
      case '~': // with type conversion
        if (t == BOOLEAN) {
          bresult = left.equals(right);
        } else if (t == STRING) {
          bresult = left.toString().equals(right.toString());
        } else if (t == INTEGER) {
          bresult = ((Number)left).longValue() == ((Number)right).longValue();
        } else if (t == FLOAT) {
          bresult = ((Number)left).doubleValue() == ((Number)right).doubleValue();
        } else {
          bresult = left.equals(right);
        }
        break;
      case '=': // strict
        bresult = left.equals(right);
        break;
```

```
        }
        result = new Boolean(bresult);
      }
      return result;
    }
    /**
     * exprRel : expr (<= >= != = expr)*
     */
    public Object exprRel() throws EventException {
      Object result = exprOrBit();
      while (token.ttype == COMPOP) {
        String opStr = token.toString(); // for debugging message.
        char type = token.ctoken;
        next();
        Object left = result;
        Object right = exprOrBit();
        boolean bresult = false;
        int t = mathType(left, right);
        if (t == STRING) {
          Object lorig = left;
          Object rorig = right;
          if (left instanceof String) {
            left = Lex.toNumber((String)left);
          }
          if (right instanceof String) {
            right = Lex.toNumber((String)right);
          }
          if (left != null && right != null) {
            t = mathType(left, right);
          } else {
            left = lorig;
            right = rorig;
          }
        }
        //System.out.println("eq or comp " + type);
        switch (type) {
        case '(':
          if (t == INTEGER) {
            bresult = ((Number)left).longValue() <= ((Number)right).longValue();
          } else if (t == FLOAT) {
            bresult = ((Number)left).doubleValue() <= ((Number)right).doubleValue();
          } else {
            throw new ExpressionException("Unexpected operand types for " +
                                          opStr + ':' +
                                          toString(left, right));
          }
          break;
        case ')':
          if (t == INTEGER) {
            bresult = ((Number)left).longValue() >= ((Number)right).longValue();
          } else if (t == FLOAT) {
            bresult = ((Number)left).doubleValue() >= ((Number)right).doubleValue();
          }
          break;
        case '<':
                                          //System.out.println("comparing .. " + left + " < " + right );
```

```
            if (t == INTEGER) {
              bresult = ((Number)left).longValue() < ((Number)right).longValue();
            } else if (t == FLOAT) {
              bresult = ((Number)left).doubleValue() < ((Number)right).doubleValue();
            } else {
              throw new ExpressionException("Unexpected operand types for " +
                                    opStr + ':' +
                                    toString(left, right));
            }
            break;
          case '>':
            //System.out.println("comparing .. " + left + " > " + right );
            if (t == INTEGER) {
              bresult = ((Number)left).longValue() > ((Number)right).longValue();
            } else if (t == FLOAT) {
              bresult = ((Number)left).doubleValue() > ((Number)right).doubleValue();
            } else {
              throw new ExpressionException("Unexpected operand types for " +
                                    opStr + ':' +
                                    toString(left, right));
            }
            break;
          }
          result = new Boolean(bresult);
        }
        return result;
      }
      private static String toString(Object left, Object right) {
        return "'" + left.toString() + "'(" + left.getClass().getName() + ") vs '" + right + "'(" + right.getClass().getName() +
')';
      }
      /**
       *  exprOrBit : exprXORBit ( | exprXORBit)*
       */
      public Object exprOrBit() throws EventException {
        Object result = exprXORBit();
        while (token.ttype == ORBITOP) {
          next();
          Object right = exprXORBit();
          Object left = result;
          if (checkIfNumber(left, right) == false) {
            throw new ExpressionException("Unexpected operand types for |");
          }
          result = new Long(((Number)left).longValue() |
                    ((Number)right).longValue());
        }
        return result;
      }
      /**
       *  exprXORBit : exprANDBit ( ^ exprANDBit)*
       */
      public Object exprXORBit() throws EventException {
        Object result = exprANDBit();
        while (token.ttype == XORBITOP) {
          char op = token.ctoken;
          next();
```

```
      Object right = exprANDBit();
      Object left = result;
      if (checkIfNumber(left, right) == false) {
        throw new ExpressionException("Unexpected operand types for " +
                                      (char)op);
      }
      result = new Long(((Number)left).longValue() ^
                        ((Number)right).longValue());
    }
    return result;
  }
  /**
   *  exprANDBit : exprShift ( & exprShift)*
   */
  public Object exprANDBit() throws EventException {
    Object result = exprShift();
    while (token.ttype == ANDBITOP) {
      char op = token.ctoken;
      next();

      Object right = exprShift();
      Object left = result;
      if (checkIfNumber(left, right) == false) {
        throw new ExpressionException("Unexpected operand types for " +
                                      (char)op);
      }
      result = new Long(((Number)left).longValue() &
                        ((Number)right).longValue());
    }
    return result;
  }
  /** evaluates part of the expression grammer. see expression.g
   * The method handles the term<BR>
   * exprTop<BR>
   *       exprShift:      exprAdd (>> exprAdd)*<BR>
   *       exprShift:      exprAdd (<< exprAdd)*<BR>
   *       ;<BR>
   */
  public Object exprShift() throws EventException {
    Object result = exprAdd();
    while (token.ttype == SHIFTOP) {
      String opStr = token.toString();
      char op = token.ctoken;
      next();

      Object right = exprAdd();
      Object left = result;
      if (checkIfNumber(left, right) == false) {
        throw new ExpressionException("Unexpected operand types for " +
                                      opStr + ':'
                                      + toString(left, right));
      }
      if (op == '>') {
        result = new Long(((Number)left).longValue() >>
                          ((Number)right).longValue());
```

```
    } else if (op == '<') {
      result = new Long(((Number)left).longValue() <<
                ((Number)right).longValue());
    } else {
      throw new ExpressionException("Unknown shift operand " + (char)op);
    }
  }
  return result;
}
/** evaluates part of the expression grammer. see expression.g
 * The method handles the term<BR>
 * exprTop<BR>
 *     :    exprMul ( (PLUS^ | MINUS^) exprMul)*<BR>
 *     ;<BR>
 */
public Object exprAdd() throws EventException {
  Object result = exprMul();
  while (token.ttype == ADDOP) {
    char op = token.ctoken;
    next();
    Object r = exprMul();
    int t = mathType(result, r);
    if (op == '+') {
      if (t != INTEGER && t != FLOAT && t != STRING) {
        throw new ExpressionException("Unexpected operand types for '"+
                          op + "' " + result.getClass() +
                          " vs " + r.getClass());
      }
      if (t == STRING) {
        result = result.toString() + r.toString();
      } else {
        Number right = (Number)r;
        Number left = (Number)result;
        if (t == INTEGER) {

          result = new Long(left.longValue() + right.longValue());
        } else if (t == FLOAT) {
          result = new Double(left.doubleValue() + right.doubleValue());
        }
      }
    } else if(op == '-') {
      if (t != INTEGER && t != FLOAT && t != STRING) {
        throw new ExpressionException("Unexpected operand types for '"+
                          op + "' " + result.getClass() +
                          " vs " + r.getClass());
      }
      Number right = (Number)r;
      Number left = (Number)result;
      if (t == INTEGER) {
        result = new Long(left.longValue() - right.longValue());
      } else if (t == FLOAT) {
        result = new Double(left.doubleValue() - right.doubleValue());
      } else if (t == STRING) {
      }
    } else {
      throw new ExpressionException("Unknown add op " + op);
```

```
    }
  }
  return result;
}
/** evaluates part of the expression grammer. see expression.g
 * The method handles the term<BR>
 * exprMul<BR>
 *     :     factor ( (STAR|DIV|MOD^) factor)*<BR>
 *     ;
 */
public Object exprMul() throws EventException {
  Object result = factor();
  while (token.ttype == MULOP) {
    char op = token.ctoken;
    next();
    Object r = factor();
    int t = mathType(result, r);
    if (t != INTEGER && t != FLOAT) {
      throw new ExpressionException("Unexpected operand types for "+
                                    op + "'");
    }

    Number right = (Number)r;
    Number left = (Number)result;
    if (op == '*') {
      if (t == INTEGER) {
        result = new Long(left.longValue() * right.longValue());
      } else if (t == FLOAT) {
        result = new Double(left.doubleValue() * right.doubleValue());
      }
    } else if(op == '/') {
      if (t == INTEGER) {
        result = new Long(left.longValue() / right.longValue());
      } else if (t == FLOAT) {
        result = new Double(left.doubleValue() / right.doubleValue());
      }
    } else if (op == '%') {
      if (t == INTEGER) {
        result = new Long(left.longValue() % right.longValue());
      } else if (t == FLOAT) {
        result = new Double(left.doubleValue() % right.doubleValue());
      }
    } else {
      throw new ExpressionException("Unknown mul op " + op);
    }
  }
  return result;
}
/** evaluates part of the expression grammer. see expression.g
 * The method handles unary ops, parenthesized expressions, as
 * defined in the term
 * <pre>
 * factor:  OPEN exprTop CLOSE
 *     |FLOAT
 *         |INTEGER
 *         |BOOLEAN
```

```
*          | STRING
*     |          unary(!-+~) atom
*     |          FUNCTION (FUNC_0, FUNC_1, FUNC_2, FUNC_N)
* </pre>
*     ;
*/
public Object factor() throws EventException {
  Object result = null;
  switch (token.ttype) {
  case OPEN:
    next();
    result = exprTop();
    if (token.ttype != CLOSE) {
      throw new ExpressionException("')' expected");
    }
    next();
    break;
  case FLOAT:
    result = new Double(token.dtoken);
    next();
    break;
  case INTEGER:
    result = new Long(token.ltoken);
    next();
    break;
  case UNDEFINED:
    result = UNDEFINED_OBJ;
    next();
    break;
  case STRING:
    result = token.stoken;
    next();
    break;
  case BOOLEAN:
    result = new Boolean(token.btoken);
    next();
    break;
  case UNOP:
    char op = token.ctoken;
    next();
    if (op == '!') {
      Boolean b = toBoolean(exprEq()); // this has priority
      result = new Boolean(! b.booleanValue());
    } else if (op == '~') {
      Number n = toNumber(exprAdd()); // until next bidop
      result = new Long(~ n.longValue());
    } else {
      throw new ExpressionException("Unexpected unop " + op);
    }
    break;
  case ADDOP:
    // sign
    char sign = token.ctoken;
    next();
    Object obj = exprOrBit();
    if (obj instanceof Number == false) {
```

```
            throw new ExpressionException("Unexpected operand " +
                                          obj.getClass().getName());
          }
          Number n = toNumber(obj); // type check
          if (sign == '-') {
            if (n instanceof Double) {
              result = new Double( - n.doubleValue());
            } else {
              result = new Long( - n.longValue());
            }
          } else {
            result = n;
          }
          break;
/*
        case CALL:
          result = call();
          break;
*/
        case FUNC_0:
          result = func0();
          break;
        case FUNC_1:
          result = func1();
          break;
        case FUNC_2:
          result = func2();
          break;
        case FUNC_N:
          result = funcN();
          break;
        case OBJECT:
          result = token.otoken;
          next();
          break;
        case ASSIGN:
          throw new ExpressionException("Assignment is not allowed : " + source);

        default:
          throw new ExpressionException("Unexpected type = " + token.ttype +
                                        " , src = " + source +
                                        " , token = " + token +
                                        " , ctoken = " + token.ctoken +
                                        " , stoken = " + token.stoken +
                                        " , ltoken = " + token.ltoken +
                                        " , dtoken = " + token.dtoken);
        }
        return result;
      }
      public Object func0() throws EventException {
        NoArgFunction f = (NoArgFunction)token.ftoken;
        //System.out.println("func = " + fname);
        next();
        if (token.ttype != OPEN) {
          throw new ExpressionException("'(' expected");
        }
```

```
    next();
    if (token.ttype != CLOSE) {
      throw new ExpressionException("')' expected");
    }
    Object result = f.perform(m_interp);
    next();
    return result;
  }

  public Object func1() throws EventException {
    UnaryFunction f = (UnaryFunction)token.ftoken;
    //System.out.println("func = " + fname);
    next();
    if (token.ttype != OPEN) {
      throw new ExpressionException("'(' expected");
    }
    next();
    Object arg = exprTop();
    if (token.ttype != CLOSE) {
      throw new ExpressionException("')' expected");
    }
    Object result = f.perform(m_interp, arg);
    next();
    return result;
  }
  public Object func2() throws EventException {
    BinaryFunction f = (BinaryFunction)token.ftoken;
    // System.out.println("func2 = " + fname);
    next();
    if (token.ttype != OPEN) {
      throw new ExpressionException("'(' expected " + token.ttype);
    }
    next();
    Object arg1 = exprTop();
    if (token.ttype != COMMA) {
      throw new ExpressionException("',' expected " + token.ttype);
    }
    next();
    Object arg2 = exprTop();
    if (token.ttype != CLOSE) {
      throw new ExpressionException("')' expected " + token.ttype);
    }
    Object result = f.perform(m_interp, arg1, arg2);
    next();
    return result;
  }
  public Object funcN() throws EventException {
    NargFunction f = (NargFunction)token.ftoken;
    next();
    if (token.ttype != OPEN) {
      throw new ExpressionException("'(' expected " + token.ttype);
    }
    next();
    Object args[] = arglist();
    if (token.ttype != CLOSE) {
      throw new ExpressionException("')' expected " + token.ttype);
```

```
    }
    Object result = f.perform(m_interp, args);
    next();
    return result;
  }
  /** evaluates part of the expression grammer. see expression.g
   * The method handles unary ops, parenthesized expressions, as
   * defined in the term<BR>
   * arglist:   exprTop ( COMMA exprTop ) * <BR>
   *      ;
   */
  public Object[] arglist() throws EventException {
    Vector args = new Vector();
    args.addElement(exprTop());
    while (token.ttype == COMMA) {
      next();
      args.addElement(exprTop());
    }
    Object arglist[] = new Object[args.size()];
    args.copyInto(arglist);
    return arglist;
  }
  static int mathType(Object left, Object right) throws ExpressionException {
    if ( (left  instanceof Double && right instanceof Number) ||
         (right instanceof Double && left  instanceof Number) ) {
      return FLOAT;
    } else if (left instanceof Number && right instanceof Number) {
      return INTEGER;
    } else if (left instanceof Boolean && right instanceof Boolean) {
      return BOOLEAN;
    } else if (left instanceof String || right instanceof String) {
      return STRING;
    } else {
      System.out.println("left = " + left.getClass().getName() + " , " +
                         "right = " + right.getClass().getName());
      return -1;
      /*
       throw new ExpressionException("Operand types mismatch " +
       left.getClass().getName() + " , " +
       right.getClass().getName());
       */
    }
  }
  boolean checkIfNumber(Object left, Object right) throws ExpressionException {
    if (left instanceof Number && right instanceof Number) {
      return true;
    }
    return false;
  }
  static Boolean toBoolean(Object obj) throws ExpressionException {
    try {
      return (Boolean)obj;
      /*
       // what if STRING, and NUMBER do we do type conversion?
       // probably yes..., but not for now. I'll do it later..
       if (obj instanceof Boolean) {
```

```
            // a bit faster
            return (Boolean)obj;
            } else if (instanceof Number) {
            return new Boolean( ((Number)obj).longValue() != 0);
            } else {
            throw new ExpressionException("Boolean expected " + obj.toString());
            }
            */
        } catch (ClassCastException ex) {
            throw new ExpressionException("Boolean expected " + obj.toString());
        }
    }
    static Number toNumber(Object obj) throws ExpressionException {
        try {
            // a bit faster
            return (Number)obj;
            // what if STRING? do we do converting?
            // probably yes..., but then do we allow this?
            // '111' + '0002' == '113', but this might be confusing
            // because
            // '0002' != '2'
            // ....
        } catch (ClassCastException ex) {
            throw new ExpressionException("INTEGER or FLOAT expected " + obj.toString());
        }
    }
}
```

```
package com.genmagic.invui.vxml.script.expr;
import com.genmagic.invui.vxml.interp.EventException;
class ExpressionException extends EventException {
   ExpressionException(String msg) {
      super("error.expression", msg);
   }
}
```

```
package com.genmagic.invui.vxml.interp;
import com.genmagic.invui.vxml.dom.Expr;
final public class ExprImpl implements Expr {
   String source;
   Object object;
   public ExprImpl(String src) throws EventException {
      source = src;
   }
   public String getExpression() {
      return source;
   }
   public Object getExpressionObject() {
      return object;
   }
   public void setExpressionObject(Object obj) {
      object = obj;
   }
   public String toString() {
      return "Expr[" + source + "]";
   }
}
```

```
package com.genmagic.invui.vxml.script.expr;
import java.util.Hashtable;
import com.genmagic.invui.vxml.lex.*;
class ExprLex extends Lex implements ExprConstants {
  static final Hashtable keywords = new Hashtable();
  static {
/*
    keywords.put("and",  new Integer(LOGAND));
    keywords.put("or",   new Integer(LOGOR));
    keywords.put("lt",   new Integer(COMPOP));
    keywords.put("gt",   new Integer(COMPOP));
    keywords.put("ltlt", new Integer(SHIFTOP));
    keywords.put("gtgt", new Integer(SHIFTOP));
    keywords.put("lshift", new Integer(SHIFTOP));
    keywords.put("rshift", new Integer(SHIFTOP));
    keywords.put("lteq", new Integer(COMPOP));
    keywords.put("gteq", new Integer(COMPOP));
*/
    // keywords.put("call", new Integer(CALL));
    keywords.put("undefined", UNDEFINED_OBJ);
    keywords.put("true", new Boolean(true));
    keywords.put("false",new Boolean(false));
    keywords.put("imax", new Long(Long.MAX_VALUE));
    keywords.put("imin", new Long(Long.MIN_VALUE));
    // should these be #xxx?
    // should we do this in string substitution?
    keywords.put("pi",   new Double(Math.PI));
    keywords.put("e",    new Double(Math.E));
    keywords.put("fmax", new Double(Double.MAX_VALUE));
    keywords.put("fmin", new Double(Double.MIN_VALUE));
    keywords.put("NaN",  new Double(Double.NaN));
    keywords.put("pinf", new Double(Double.POSITIVE_INFINITY));
    keywords.put("ninf", new Double(Double.NEGATIVE_INFINITY));
    BuiltInFunctions.init(keywords);
  }
  ExprLex(String s) {
    super(s, new ExprToken());
  }
  protected int handleRawString() throws LexException {
    // check keyword
    if (keywords == null) {
      token.variable = token.stoken;
      return VARIABLE;
    }
    Object o = keywords.get(token.stoken);
    if (o == null) {
      token.variable = token.stoken;
      return VARIABLE;
    }
    if (o instanceof Integer) {
      int c = ((Integer)o).intValue();
      switch (c) {
      case COMPOP:
        if ("lt".equals(token.stoken)) {
          token.ctoken = '<';
        } else if ("gt".equals(token.stoken)) {
```

```
            token.ctoken = '>';
          } else if ("lteq".equals(token.stoken)) {
            token.ctoken = '(';
          } else if ("gteq".equals(token.stoken)) {
            token.ctoken = ')';
          } else {
            throw new LexException("Unexpected COMPOP :"+token.stoken);
          }
          break;
        case SHIFTOP:
          if (token.stoken.charAt(0) == 'l' &&
            ("ltlt".equals(token.stoken) ||
             "lshift".equals(token.stoken))) {
            token.ctoken = '<';
          } else if ("gtgt".equals(token.stoken) ||
                "rshift".equals(token.stoken)) {
            token.ctoken = '>';
          }
          break;
        case LOGAND:
        case LOGOR:
          break;
          //return c;
        }
        return c;
      } else if (o instanceof Boolean) {
        token.btoken = ((Boolean)o).booleanValue();
        return BOOLEAN;
      } else if (o instanceof Double) {
        token.dtoken = ((Double)o).doubleValue();
        return FLOAT;
      } else if (o instanceof Long) {
        token.ltoken = ((Long)o).longValue();
        return INTEGER;
      } else if (o instanceof NoArgFunction) {
        ((ExprToken)token).ftoken = (Function)o;
        return FUNC_0;
      } else if (o instanceof UnaryFunction) {
        ((ExprToken)token).ftoken = (Function)o;
        return FUNC_1;
      } else if (o instanceof BinaryFunction) {
        ((ExprToken)token).ftoken = (Function)o;
        return FUNC_2;
      } else if (o instanceof NargFunction) {
        ((ExprToken)token).ftoken = (Function)o;
        return FUNC_N;
      } else if (o == UNDEFINED_OBJ) {
        return UNDEFINED;
      } else {
        throw new LexException("Unexpected value : " + token.stoken);
      }
    }
  }
```

```
package com.genmagic.invui.vxml.script.expr;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.Promptable;
import com.genmagic.invui.vxml.lex.*;
final class ExprToken extends Token implements ExprConstants {
  Function ftoken;
  ExprToken evaluate(VXMLInterp interp) throws EventException {
    if (ttype != VARIABLE) {
      return this;
    }
    ExprToken t = (ExprToken)clone();
    Object obj = interp.getVariable(variable);
    if (obj == null) {
      t.stoken = "";
      t.ttype = STRING;
      return t;
    }
    if (obj instanceof Boolean) {
      t.btoken = ((Boolean)obj).booleanValue();
      t.ttype = BOOLEAN;
      return t;
    } else if (obj instanceof Long || obj instanceof Integer) {
      t.ltoken = ((Number)obj).longValue();
      t.ttype = INTEGER;
      return t;
    } else if (obj instanceof Double || obj instanceof Float) {
      t.dtoken = ((Number)obj).doubleValue();
      t.ttype = FLOAT;
      return t;
    } else if (obj instanceof String) {
      t.stoken = (String)obj;
      t.ttype = STRING;
      return t;
    } else if (obj == UNDEFINED_OBJ) {
      t.ttype = UNDEFINED;
      return t;
    } else if (obj instanceof Promptable) {
      t.ttype = OBJECT;
      t.otoken = obj;
      return t;
    } else {
      System.out.println("UNKNOWN " + obj.getClass());
    }
    return t;
  }
}
```

```
package com.genmagic.invui.vxml.dom;
import com.genmagic.invui.vxml.interp.VXMLConstants;
/*
 * @since Vxml1.0
 */
public interface Fetchable extends VXMLConstants {
/*
   static public final int SAFE = 0;
   static public final int FAST = 1;
*/
   public int getCaching();
   public int getFetchTimeout();
   public int getFetchHint();
}
```

```
package com.genmagic.invui.vxml.domimpl;
import com.genmagic.invui.vxml.interp.Util;
import com.genmagic.invui.vxml.interp.VXMLConstants;
import com.genmagic.invui.vxml.interp.RuntimeEventException;
import com.genmagic.invui.vxml.dom.VXMLElement;
class FetchInfo implements VXMLConstants {
   int caching = -1;
   int fetchTimeout = -1;
   int fetchHint = -1;
   FetchInfo(VXMLElement e) {
      String cachingStr = e.getAttribute2("caching");
      String fetchHintStr = e.getAttribute2("fetchhint");
      if (cachingStr != null) {
         if ("safe".equals(cachingStr)) {
            caching = SAFE;
         } else if ("fast".equals(cachingStr)) {
            caching = FAST;
         } else {
            throw new RuntimeEventException("error.semantic",
                                 "invalid caching :" +
                                 caching);
         }
      }
      fetchTimeout = Util.getTimeAttribute(e, "fetchtimeout", -1);
      if (fetchHintStr != null) {
         if ("prefetch".equals(fetchHintStr)) {
            fetchHint = PREFETCH;
         } else if ("safe".equals(fetchHintStr)) {
            fetchHint = SAFE;
         } else if ("stream".equals(fetchHintStr)) {
            fetchHint = STREAM;
         } else {
            throw new RuntimeEventException("error.semantic",
                                 "invalid fetchhint :" +
                                 fetchHintStr);
         }
      }
   }
}
```

```
package com.genmagic.invui.vxml.dom;
import org.w3c.dom.Node;
import com.genmagic.invui.vxml.interp.Grammar;
import com.genmagic.invui.vxml.interp.GrammarSet;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
abstract public class Field extends FormItem {
   protected Field() {
      super("field");
   }
   protected Field(String name) {
      super(name);
   }
   /**
    * Type of guard variable
    */
   abstract public String getType();
   /**
    * Type of guard variable
    */
   abstract public boolean getModal();
   /**
    * @deprecated as of Vxml1.0
    */
   abstract public int getTimeout();
   /**
    * @deprecated as of Vxml1.0
    */
   abstract public int getToomuchTimeout();
   /**
    * @deprecated as of Vxml1.0
    */
   abstract public int getEndspeechTimeout();
   /**
    * Slot Name
    */
   abstract public String getSlot();
   /**
    * Returns prompt related children only.
    * prompt, audio, PCDATA, value, enumerate
    */
   abstract public Node[] getPromptElements();
   /**
    * Get active grammar definitions on this form item.
    */
   abstract public com.genmagic.invui.vxml.interp.GrammarSet getActiveGrammars(VXMLInterp ip) throws
EventException;
   /**
    *
    */
   abstract public com.genmagic.invui.vxml.interp.Grammar[] getGrammars(VXMLInterp ip) throws EventException;
   /*
   public Filled[] getFilled(String varNames[], VXMLInterp ip) {
      Filled filled[] = getFilled();
      if (filled != null) {
         java.util.Vector v = new java.util.Vector();
```

```
        for (int i=0; i<filled.length; i++) {
            if (filled[i].match (varNames, ip)) {
                v.addElement(filled[i]);
            }
        }
        filled = new Filled[v.size()];
        v.copyInto(filled);
    }
    return filled;
  }
  */
}
```

```
package com.genmagic.invui.vxml.tags;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.dom.Field;
import com.genmagic.invui.vxml.dom.Catch;
import com.genmagic.invui.vxml.dom.Filled;
import com.genmagic.invui.vxml.interp.*;
public class FieldTag implements TagHandler, VXMLConstants {
  public void perform(VXMLInterp ip, Element self) throws VXMLException {
    Field field = (Field)self;
    SpeechControl speech = ip.getSpeechControl();
    GrammarSet grammarSet = field.getActiveGrammars(ip);
    XMLLoader loader = ip.getAppContext().getLoader();
    String stateName =
      loader.getDocumentBase() +
      loader.getFile() + "#" +
      field.getStateName();
    // speech.setReprompt(true);
    // speech.backupPrompts();
    String varName = field.getName();
    Properties props = ip.getCurrentProperties();
    try {
      speech.enterState(stateName);
      if (speech.getReprompt()) {
        ip.eval(field.getPromptElements());
        speech.setReprompt(false);
      }
      // get property after we evaluated prompts
      int timeout = speech.getTimeout();
      if (timeout<0) {
        timeout = (int)props.getTimeProperty("timeout",
                                DEFAULT_REC_TIMEOUT);
      }
      speech.clearTimeout();

      int toomuch = (int)props.getTimeProperty("toomuchtimeout",
                                DEFAULT_TOOMUCH_TIMEOUT);

      int endspeech = (int)props.getTimeProperty("endspeechtimeout",
                                  DEFAULT_ENDSPEECH_TIMEOUT);
      RecResult result =
        speech.playAndRecognition(grammarSet,
                        timeout,
                        toomuch,
                        endspeech,
                        DEFAULT_MAX_DIGIT);
      // if any
      result.performAction(ip);
      SlotResult sr = result.getResult(field.getSlot());
      if (sr == null) {
        throw new EventException("nomatch");
      }
      ip.debug("set var name=" + varName + " val=" + sr.value);
      ip.setVariable(varName, sr.value);
      ip.createVariable(varName + "$.confidence",
                new Double(sr.confidence));
```

```
            ip.createVariable(varName + "$.utterance",
                              sr.utterance);
            Filled filled[] = field.getParentForm().getFilled(new
String[]{varName}, ip);
            if (filled != null) {
               ip.eval(filled);
            }
         } catch(VXMLException ex) {
            throw ex;
         } catch(RuntimeException ex) {
            throw ex;
         } catch (Exception ex) {
            ex.printStackTrace();
            throw new EventException("error.internal", ex.getMessage());
         } finally {
            speech.leaveState();
         }
      }
            }
```

```
package com.genmagic.invui.vxml.dom;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLConstants;
import com.genmagic.invui.vxml.interp.EventException;
abstract public class Filled extends VXMLElement {
   public Filled() {
      super("filled");
   }
   static public final boolean ANY = true;
   static public final boolean ALL = false;
   abstract public String[] getNameList();
   abstract public boolean getMode();
   private boolean checkVarNames(String varNames[], String[] namelist) {
      for (int i=0; i<varNames.length; i++) {
         for (int j=0; j<namelist.length; j++) {
            if (varNames[i].equals(namelist[j])) {
               return true;
            }
         }
      }
      return false;
   }
   public boolean match (String varNames[], VXMLInterp ip) {
      String[] namelist = getNameList();
      if (checkVarNames(varNames, namelist)==false) {
         return false;
      }
      boolean mode = getMode();
      if (mode == ALL) {
         for(int i=0; i<namelist.length; i++) {
            if (ip.hasDefinedVariable(VXMLConstants.LOCAL, namelist[i])
               == false) {
               return false;
            }
         }
         return true;
      } else {
         for(int i=0; i<namelist.length; i++) {
            if (ip.hasDefinedVariable(VXMLConstants.LOCAL, namelist[i])) {
               return true;
            }
         }
         return false;
      }
   }
}
```

```
package com.genmagic.invui.vxml.tags;
public class FilledTag extends ExecutableTag {
}
```

Oracle Ex. 2010, p. 1370

```
package com.genmagic.invui.vxml.dom;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLConstants;
import com.genmagic.invui.vxml.interp.GrammarSet;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.dom.VXMLElement;
import java.util.Vector;
abstract public class Form extends Dialog {
  public Form() {
    super("form");
  }
  /**
   * Var elements in this form
   * @deprecated as of VXML1.1
   */
  abstract public Var[] getVarDecls();
  /**
   * Get declarations
   * This includes FormItems and caller should check if it's
   * FormItem or not.
   * @since as of VXML1.1
   */
  abstract public VXMLElement[] getDecls();
  /**
   * Get all form items
   */
  abstract public FormItem[] getFormItems();
  /**
   * Get initial form item.
   */
  abstract public Initial getInitial();
  /**
   * Get active grammar definitions on this form item.
   */
  // abstract public com.genmagic.invui.vxml.interp.Grammar[] getFormGrammars();
  abstract public void getFormGrammars(VXMLInterp ip, GrammarSet set) throws EventException ;
  /**
   * This should keep document order.
   */
  abstract public Filled[] getFilled();
  /**
   *
   */
  public Filled[] getFilled(String varNames[], VXMLInterp ip) {
    Filled filled[] = getFilled();
    if (filled != null) {
      java.util.Vector v = new java.util.Vector();
      for (int i=0; i<filled.length; i++) {
        if (filled[i].match (varNames, ip)) {
          v.addElement(filled[i]);
        }
      }
      filled = new Filled[v.size()];
      v.copyInto(filled);
    }
    return filled;
```

```
    }
    /**
     *
     */
    public String[] getAllGuardVariables() {
        FormItem[] items = getFormItems();
        String names[] = new String[items.length];
        for (int i=0; i<items.length; i++) {
            names[i] = items[i].getName();
        }
        return names;
    }
    /**
     *
    public String[] getAllSlotNames() {
        FormItem[] items = getFormItems();
        Vector v = new Vector();
        for (int i=0; i<items.length; i++) {
            FormItem item = items[i];
            if (item instanceof Field) {
                v.addElement(((Field)item).getName());
            }
        }
        String names[] = new String[v.size()];
        v.copyInto(names);
        return names;
    }
    */
    public FormItem getFormItem(String name) {
        FormItem[] items = getFormItems();
        for (int i=0; i<items.length; i++) {
            if (name.equals(items[i].getName())) {
                return items[i];
            }
        }
        return null;
    }
    /**
     *
     */
    abstract public FormItem getNextFormItem(VXMLInterp ip) throws EventException;
}
```

```
package com.genmagic.invui.vxml.dom;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.EventCounter;
import com.genmagic.invui.vxml.interp.NotImplementedException;
abstract public class FormItem extends VXMLElement {
  public FormItem(String tagname) {
    super(tagname);
  }
  /*
   * Gets event handler (catch)
   */
  abstract public Catch getEventHandler(VXMLInterp ip, Exception ev, int count) throws EventException ;
  /**
   * Var name
   */
  abstract public String getName();
  /**
   * Initial value
   * @since Vxml1.0
   */
  abstract public Expr getExpr() throws EventException;
  /**
   * Guard Condition
   * @since Vxml1.0
   */
  abstract public Expr getCond() throws EventException;
  /**
   * State Name
   */
  abstract public String getStateName();
  /**
   *
   */
  public EventCounter getEventCounter() {
    throw new NotImplementedException("getEventCounter is not available");
  }
  /**
   * Set Default Guard Variable.
   * This is for 0.9. For 1.0 implementation, please see
   */
  public void setDefaultGuardVariable(VXMLInterp ip) throws EventException {
    ip.createVariable(getName());
  }
  abstract public Property[] getPropertyDecls();
  abstract public void getFilled(java.util.Vector v);
}
```

```
package com.genmagic.invui.vxml.domimpl;
import java.util.Vector;
import org.w3c.dom.Node;
import com.genmagic.invui.vxml.interp.Util;
import com.genmagic.invui.vxml.interp.VXMLDocumentImpl;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.EventCounter;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.dom.VXMLElement;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.dom.Dialog;
import com.genmagic.invui.vxml.dom.FormItem;
import com.genmagic.invui.vxml.dom.Property;
class FormItemInfo {
   private FormItem e;
   EventCounter counter;
   String name;
   String stateName;
   Property[] propertyDecls = null;
   private Expr expr = null;
   private Expr cond = VXMLElement.NULL_EXPR;
   FormItemInfo(FormItem e, String type) {
      this.e = e;
      name = e.getAttribute2("name");
      if (name == null) {
         name = VXMLDocumentImpl.genVarName(type);
      }
      Dialog d = (Dialog)e.getParentNode();
      stateName = d.getID() + ":" + name;
      Vector v = new Vector();
      for (Node child = e.getFirstChild();
          child != null;
          child = child.getNextSibling()) {
         if (child instanceof Property) {
            v.addElement(child);
         }
      }
      propertyDecls = new Property[v.size()];
      v.copyInto(propertyDecls);
   }
   Expr getExpr() throws EventException {
      if (expr == null) {
         expr = Util.getExprAttribute(e, "expr");
      }
      return expr;
   }
   Expr getCond() throws EventException {
      if (cond == VXMLElement.NULL_EXPR) {
         cond = Util.getExprAttribute(e, "cond");
      }
      return cond;
   }
   /**
    * remind: This may need to move to tag handler.
    */
   EventCounter getEventCounter() {
```

```
    if (counter == null) {
      counter = new EventCounter();
    }
    return counter;
  }
  void setDefaultGuardVariable(VXMLInterp ip) throws EventException {
    Expr expr = getExpr();
    if (expr == null) {
      ip.createVariable(e.getName());
    } else {
      ip.createVariable(e.getName(),
                ip.evalExpression(expr));

    }
  }
}
```

```
package com.genmagic.invui.vxml.tags;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.dom.*;
import com.genmagic.invui.vxml.interp.*;
public class FormTag implements TagHandler {
  public void perform(VXMLInterp ip, Element self) throws VXMLException {
    Form form = (Form)self;
    VXMLElement[] decls = form.getDecls();
    for (int i=0; i<decls.length; i++) {
      VXMLElement decl = decls[i];
      if (decl instanceof FormItem) {
        FormItem item = (FormItem)decl;
        item.setDefaultGuardVariable(ip);
        item.getEventCounter().reset();
      } else {
        ip.eval(decl);
      }
    }
    // ip.eval(form.getVarDecls());
    RecResult res = ip.getInitialDialogRecResult();
    VXMLElement[] handlers = null;
    if (res != null) {
      FormItem[] items = form.getFormItems();
      String[] updated = Util.copyVariablesFromRecResult(ip, items, res);
      handlers = form.getFilled(updated, ip);
    }
    FormItem currentItem = null;
    FormItem nextItem = null;
    EventCounter counter = null;
    SpeechControl speech = ip.getSpeechControl();
    boolean scopeCreated = false;
    try {
      while(true) {

        try {
          if (handlers != null) {
            VXMLElement[] h = handlers;
            handlers = null;
            ip.eval(h);
            continue;
          }

          if (nextItem == null) {
            nextItem = form.getNextFormItem(ip);
          }
          if (nextItem == null) {
            throw new EventException("exit");
          }

          if (currentItem != nextItem) {
            counter = nextItem.getEventCounter();
            if (scopeCreated) {
              ip.endPropertyScope();
            }
            ip.beginPropertyScope();
            scopeCreated = true;
```

```
        // block should not have property decls
        ip.eval(nextItem.getPropertyDecls());

        // reset prompt counter
        // promptCount = 1;
        // keep the prompts accumulated in previous
        // field or form.
        speech.backupPrompts();
        // play prompts
        speech.setReprompt(true);
      } else {
        if (speech.getReprompt()) {
          speech.backupPrompts();
          counter.incrementPromptCounter();
        }
      }

      speech.setCount(counter.getPromptCount());

      currentItem = nextItem;
      nextItem = null;
      ip.diag("exeuting form item " + currentItem.getName());
      ip.eval(currentItem);

    } catch (GotoNextItemException ex) {
      nextItem = ex.getNextItem();
      if (nextItem != null) {
        ip.clearVariable(VXMLConstants.DIALOG, nextItem.getName());
      }
    } catch (EventException ex) {
      String name = currentItem != null ?
        "field:" + currentItem.getName() :
        ("form:" + form.getID());
      String eventName = ex.getEventName();
      ip.diag("event exception in :" + name +  " event=" +
          eventName + ", msg=" + ex.getMessage());
      Catch c = null;
      if (counter != null) {
        c = currentItem.getEventHandler(ip,
                                ex,
                                counter.getCount(eventName));
      } else {
        c = form.getEventHandler(ip, ex, 1);
      }
      if (c == null) {
        ip.diag("handler not found for " + eventName);
        throw ex;
      }
      handlers = new VXMLElement[1];
      handlers[0] = c;
    } catch (RuntimeEventException ex) {
      String name = currentItem != null ?
        "field:" + currentItem.getName() :
        ("form:" + form.getID());
      String eventName = ex.getEventName();
      ip.diag("event exception in :" + name + " event=" +
```

```
                    eventName + ", msg=" + ex.getMessage());
                Catch c = null;
                if (counter != null) {
                  c = currentItem.getEventHandler(ip,
                                  ex,
                                  counter.getCount(eventName));
                } else {
                  c = form.getEventHandler(ip, ex, 1);
                }
                if (c == null) {
                  ex.printStackTrace();
                  ip.diag("handler not found for " + eventName);
                  throw ex;
                }
                handlers = new VXMLElement[1];
                handlers[0] = c;
              } catch(VXMLException ex) {
                throw ex;
              } catch(RuntimeException ex) {
                throw ex;
              } catch(Exception ex) {
                String name = currentItem != null ?
                  "field:" + currentItem.getName() :
                  ("form:" + form.getID());
                String eventName = "error.internal";
                ip.diag("event exception in :" + name + " event=" +
                    eventName + ", msg=" + ex.getMessage());
                Catch c = null;
                if (counter != null) {
                  c = currentItem.getEventHandler(ip,
                                  ex,
                                  counter.getCount(eventName));
                } else {
                  c = form.getEventHandler(ip, ex, 1);
                }
                if (c == null) {
                  ex.printStackTrace();
                  ip.diag("handler not found for " + eventName);
                  throw new EventException(eventName,
                              ex.getMessage());
                }
                handlers = new VXMLElement[1];
                handlers[0] = c;
              }
            }
          } finally {
            ip.endPropertyScope();
          }
        }
      }
```

```
package com.genmagic.invui.vxml.script;
import java.util.Hashtable;
public interface Frame {
   public Object getVariable(String name);
   public java.util.Enumeration getVariableNames();
}
```

Oracle Exhibit 2010

```
package com.genmagic.invui.vxml.script.expr;
/** interface for function in expressions, this may not be necessary..
 * @see Expression
 */
interface Function extends com.genmagic.invui.vxml.lex.LexConstants {
}
```

Document Page 2 of 2

```
package com.genmagic.invui.vxml.dom;
import com.genmagic.invui.vxml.interp.EventException;
abstract public class Goto extends VXMLElement implements DocumentFetchable {
   public Goto() {
      super("goto");
   }
   /**
    * Get next item
    */
   abstract public String getNextItem();
   /**
    * Get next item
    * @since Vxml1.0
    */
   abstract public Expr getExprItem() throws EventException;
   /**
    * Get next
    */
   abstract public String getNext() throws EventException;
   /**
    * Get next
    * @since Vxml1.0
    */
   abstract public Expr getExpr() throws EventException;
   /**
    * @deprecated as of Vxml1.0
    */
   abstract public String[] getSubmit();
   /**
    * @deprecated as of Vxml1.0
    */
   abstract public int getMethod();
   /**
    * @deprecated as of Vxml1.0
    */
   abstract public String getEnctype();
   /**
    * moved to Fetchable
   abstract public int getCaching();
    */
   /**
    * moved to Fetchable
   abstract public int getFetchTimeout();
    */
}
```

```
package com.genmagic.invui.vxml.interp;
import com.genmagic.invui.vxml.dom.Dialog;
public class GotoNextException extends VXMLException implements VXMLConstants {
   private Dialog nextDialog = null;
   private RecResult result = null;
   private int mode = EXTERNAL;
   public GotoNextException(Dialog n, int m) {
      nextDialog = n;
      mode = m;
   }
   public GotoNextException(Dialog n, RecResult r, int m) {
      nextDialog = n;
      mode = m;
      result = r;
   }
   public Dialog getNextDialog() {
      return nextDialog;
   }
   public int getMode() {
      return mode;
   }
   public RecResult getRecResult() {
      return result;
   }
}
```

```
package com.genmagic.invui.vxml.interp;
import com.genmagic.invui.vxml.dom.FormItem;
public class GotoNextItemException extends VXMLException {
   private FormItem next = null;
   private RecResult result = null;
   public GotoNextItemException(FormItem n) {
      next = n;
   }
   public GotoNextItemException(RecResult r) {
      result = r;
   }
   public GotoNextItemException() {
   }
   public FormItem getNextItem() {
      return next;
   }
   public RecResult getRecResult() {
      return result;
   }
}
```

Document Page 2 of 2

```
package com.genmagic.invui.vxml.tags;
import java.lang.Object;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.dom.Goto;
import com.genmagic.invui.vxml.dom.Form;
import com.genmagic.invui.vxml.dom.Dialog;
import com.genmagic.invui.vxml.dom.FormItem;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLConstants;
import com.genmagic.invui.vxml.interp.VXMLException;
import com.genmagic.invui.vxml.interp.TagHandler;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.GotoNextException;
import com.genmagic.invui.vxml.interp.GotoNextItemException;
import com.genmagic.invui.vxml.interp.SpeechControl;
import com.genmagic.invui.vxml.prompt.CompositePrompt;
public class GotoTag implements TagHandler, VXMLConstants {
  public void perform(VXMLInterp ip, Element self) throws VXMLException {
    Goto gt = (Goto)self;
    String nextItem = gt.getNextItem();
    if (nextItem == null) {
      Expr expr = gt.getExprItem();
      if (expr != null) {
        nextItem = ip.evalExpression(expr).toString();
      }
    }
    if (nextItem != null) {
      Dialog d = ip.getCurrentDialog();
      if (d instanceof Form == false) {
        throw new EventException("error.internal",
                    "form not found");
      }
      FormItem item = ((Form)d).getFormItem(nextItem);
      if (item == null) {
        throw new EventException("error.badnext",
                    "nextitem not found " + nextItem);
      }
      throw new GotoNextItemException(item);
    }
    String next = gt.getNext();
    if (next == null) {
      Expr expr = gt.getExpr();
      if (expr != null) {
        next = ip.evalExpression(expr).toString();
      }
    }
    if (next == null) {
      throw new EventException("error.semantic",
                  "no next info in goto");
    }
    // fetch hint
    // fetch audio
    SpeechControl speech = ip.getSpeechControl();
    CompositePrompt backup = null;
```

```
    String str = gt.getAttribute2("fetchAudioBreak");
    boolean breakFetchAudio = str != null ? Boolean.valueOf(str).booleanValue() : true;
    String src = gt.getFetchAudio();
    if(src != null)
    {
      backup = speech.getCompositePrompt();
      speech.clearPrompts();
      src = ip.loadAudio(src,
            gt.getCaching(),
            gt.getFetchTimeout());
      speech.appendAudio(src, "");
      speech.playPrompt2(true); // async play prompts
    }
    else
    {
      if (gt.hasChildNodes())
      {
        backup = speech.getCompositePrompt();
        speech.clearPrompts();
        ip.evalChildren(gt);
        speech.getCompositePrompt();
        speech.playPrompt2(true); // async play prompts
      }
    }
    try
    {
      ip.gotoNext(next,
          gt.getCaching(),
          gt.getFetchTimeout());
/*
          gt.getSubmit(),
          gt.getMethod(),
          gt.getEnctype(),
          gt.getCaching(),
          gt.getFetchTimeout());
*/
    } catch(VXMLException ex) {
      throw ex;
    } finally {
      if(backup != null)
      {
        speech.stopPlay(!breakFetchAudio);
        speech.clearPrompts();
        // restore old prompt
        backup.appendTo(speech);
      }
    }
  } // perform
};
```

```
package com.genmagic.invui.vxml.dom;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
abstract public class Grammar extends VXMLElement implements Fetchable {
   public Grammar() {
      super("grammar");
   }
   public Grammar(String name) {
      super(name);
   }
   abstract public String getSrc();
   abstract public int getScope();
   abstract public String getType();
   abstract public com.genmagic.invui.vxml.interp.Grammar getGrammar(VXMLInterp ip) throws EventException;
}
```

Document 3 - Page 5 of 5

```
package com.genmagic.invui.vxml.interp;
/*
import com.genmagic.mts.Session;
*/
import com.genmagic.invui.vxml.dom.VXMLElement;
import com.genmagic.invui.vxml.idl.invui.*;
import com.genmagic.invui.vxml.idl.invui.SlotPackage.*;
import org.omg.CORBA.UserException;
abstract public class Grammar {
  /**
   * Default Grammar Action, which does nothing.
   */
  static private GrammarAction defaultAction = new GrammarAction() {
    public void performAction(Grammar g, RecResult r, Object cxt) {
    }
  };
  abstract public String getName();
  abstract public String getUniqID();
  abstract public String[] getSlotNames();
  abstract public float getWeight();
  public void setAction(GrammarAction a) {
    action = a;
  }
  public void performAction(RecResult result, Object cxt) throws Exception {
    action.performAction(this, result, cxt);
  }
  public SlotResult getSlotResult(RecResult result, String name) throws EventException {
    return result.getRawResult(name);
  }
  public Grammar getActiveGrammar(RecResult rec) throws EventException {
    return this;
  }
  protected GrammarAction action = defaultAction;
  /**
   * For CORBA version
   */
  GrammarHandle getGrammarHandle(VUISession session) throws UserException, EventException {
    GrammarHandle h = new GrammarHandle();
    h.name(getName());
    return h;
  }
  public PlayAndRecResults recognize(VUISession session,
                                     int rec_timeout,
                                     int toomuch_timeout,
                                     int endspeech_timeout)
     throws UserException, EventException
  {
    PlayAndRecResults result =
      session.playAndRecognize(getGrammarHandle(session),
                               rec_timeout,
                               toomuch_timeout,
                               endspeech_timeout);
    return result;
  }
  /**
   * FOR CMS API
```

```
   */
/*
   public void startRecognition(Session session,
                        int rec_timeout,
                        int toomuch_timeout,
                        int endspeech_timeout,
                        int max_digit) throws com.genmagic.mts.MTSException
   {
      session.recognition(getName(),
                  rec_timeout,
                  toomuch_timeout,
                  endspeech_timeout,
                  max_digit);
   }
*/
   public String toString() {
      return "Grammar[name=" + getName() + ",id="+getUniqID() +",action="+
         action + "]";
   }
}
```

```
package com.genmagic.invui.vxml.interp;
public interface GrammarAction {
   public void performAction(Grammar gram, RecResult result, Object context) throws Exception;
}
```

Petitioner's Exhibit 1008, p. 1008

```
package com.genmagic.invui.vxml.property;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.VXMLException;
public class GrammarListPropertyListener implements PropertyChangeListener {
  public void propertyAdded(VXMLInterp ip, Object value) throws VXMLException {
    ip.getSpeechControl().setParameter("jde.grammarlist",
                                       value.toString());
  }
  public void propertyRemoved(VXMLInterp ip) throws VXMLException {
    String previous =
      ip.getCurrentProperties().getProperty("grammarlist",
                                            null);
    ip.getSpeechControl().setParameter("jde.grammarlist",
                                       previous);
  }
};
```

```
package com.genmagic.invui.vxml.interp;
/*
import com.genmagic.mts.Session;
import com.genmagic.mts.MTSException;
*/
import java.util.Hashtable;
import java.util.Vector;
import java.util.Enumeration;
import org.omg.CORBA.UserException;
import com.genmagic.invui.vxml.idl.invui.*;
import com.genmagic.invui.vxml.idl.invui.DynamicGrammarPackage.*;
public class GrammarSet extends DynamicGrammar {
   private String _key = null;
   private String _activeGrammarNames = null;
   private Hashtable _grammarTable = new Hashtable();
   private GrammarHandle _handle = null;
   public String getKey() {
      if (_key == null) {
         // synchronized
         _key = genKey(getActiveGrammarNames());
      }
      return _key;
   }
   public float getWeight() {
      return -1.0f; // no weight for grammar set
   }
   public String getName() {
      return DYNAMIC_TOP;
   }
   public String getActiveGrammarNames() {
      if (_activeGrammarNames == null) {
         StringBuffer buf = new StringBuffer("[ ");
         Enumeration e = _grammarTable.elements();
         while (e.hasMoreElements()) {
            Grammar g = (Grammar)e.nextElement();
            if (g instanceof DynamicGrammar) {
               buf.append("DYNAMIC_TEXT(" + g.getUniqID() + ")");
            } else {
               buf.append(g.getName());
            }
            float weight = g.getWeight();
            if (weight > 0.0) {
               buf.append("*" + weight + " ");
            } else {
               buf.append(" ");
            }
         }
         buf.append("]");
         _activeGrammarNames = buf.toString();
      }
      return _activeGrammarNames;
   }
   public String[] getSlotNames() {
      throw new RuntimeException("GrammarSet does not have slots");
   }
   public boolean equals(Object obj) {
```

```
    if (obj instanceof GrammarSet) {
      GrammarSet gs = (GrammarSet)obj;
      return gs._grammarTable.equals(_grammarTable);
/*
      if (gs._grammars.size() == _grammars.size()) {
        for (int i=0; i<_grammars.size(); i++) {
          // the order is important?
          if (_grammars[i].equals(gs._grammars[i])==false) {
            return false;
          }
        }
        return true;
      }
      //return getName().equals(gs.getName());
*/
    }
    return false;
  }
  /**
   * FOR CMS API
   */
/*
  private String _dynamicGrammarSrc = null;
  private String getSrc() {
    if (_dynamicGrammarSrc == null) {
      StringBuffer buf = new StringBuffer("[ ");
      int dyn = 0;
      Enumeration e = _grammarTable.elements();
      while (e.hasMoreElements()) {
        Grammar g = (Grammar)e.nextElement();
        if (g instanceof DynamicGrammar) {
          buf.append(dynamicGrammarRefs[dyn++] +
                " {<namespace \"" + g.getUniqID() + "\">} ");
        } else {
          buf.append(g.getName() +
                " {<namespace \"" + g.getUniqID() + "\">} ");
        }
      }
      buf.append("]");
      _dynamicGrammarSrc = buf.toString();
    }
    return _dynamicGrammarSrc;
  }
  public void install(Session sess) throws MTSException {
    Enumeration e = _grammarTable.elements();
    while (e.hasMoreElements()) {
      Object g = e.nextElement();
      if (g instanceof DynamicGrammar) {
        ((DynamicGrammar)g).install(sess);
      }
    }
    String key = getKey();
    if (sess.hasDynamicGrammar(key) == false) {
      sess.createDynamicGrammar(key, getSrc());
      int stat = sess.waitDynamicGrammarCreation();
      if (stat != DG_Success) {
```

```
            throw new MTSException("Fail to create dynamic grammar :" +
                        DG_STATUS_STRING[stat]);
          }
        }
      }
    */
      /**
       * FOR CMS API
       */
    /*
      public void insertGrammar(Session session) throws MTSException {
        int dyn = 0;
        Enumeration e = _grammarTable.elements();
        while (e.hasMoreElements()) {
          Grammar g = (Grammar)e.nextElement();
          if (g instanceof DynamicGrammar) {
            _grammarTable.put(g.getUniqID(), g);
            DynamicGrammar dg = (DynamicGrammar)g;
            session.insertDynamicGrammar(dg.getKey(),
                        dynamicGrammarNames[dyn++]);
          } else {
            _grammarTable.put(g.getUniqID(), g);
          }
        }
        session.insertDynamicGrammar(getKey(), DYNAMIC_ROOT);;
      }
    */
      /**
       * gets active grammar for this recognition results
       */
      public Grammar getActiveGrammar(RecResult rec) throws EventException {
        if (rec.getNumSlots() == 0) {
          throw new EventException("nomatch");
        }
        Grammar g = (Grammar)_grammarTable.get(rec.getActiveGrammarName());
        if (g == null) {
          throw new EventException("nomatch");
        }
        return g;
      }
      /**
       * For CORBA API
       */
      GrammarHandle getGrammarHandle(VUISession session) throws UserException, EventException {
        if (_handle == null) {
          com.genmagic.invui.vxml.idl.invui.DynamicGrammar gram =
            new com.genmagic.invui.vxml.idl.invui.DynamicGrammar();
          GrammarItem[] items = new GrammarItem[_grammarTable.size()];
          int i= 0;
          Enumeration e = _grammarTable.elements();
          while (e.hasMoreElements()) {
            Grammar grammar = (Grammar)e.nextElement();
            // System.out.println("getting grammar handle for " + grammar);
            GrammarItem item = items[i] = new GrammarItem();
            item.handle = grammar.getGrammarHandle(session);
            item.weight = grammar.getWeight();
```

```
            i++;
          }
          gram.set(items);
          try {
            _handle = session.getGrammarHandle(gram);
          } catch (SpeechError ex) {
            throw new EventException("error.mts.speech",
                                     "could not compile grammar set : error_code="
                                     + ex.error_code);
          }
        }
        return _handle;
      }
      public void addGrammars(Grammar[] _grammars) throws EventException {
        for (int i=0; i<_grammars.length; i++) {
          addGrammar(_grammars[i]);
        }
      }
      public int size() {
        return _grammarTable.size();
      }
      public void addGrammar(Grammar gram) {
        if (gram instanceof NoneGrammar) {
          return;
        } else if (_grammarTable.containsKey(gram.getUniqID()) == false) {
          // System.out.println("adding.. " + gram);
          _grammarTable.put(gram.getUniqID(), gram);
        } else {
          // System.out.println("******* Skipping grammar.. " + gram);
        }
      }
    /*
      public void resolve(VXMLInterp ip) throws EventException {
        Enumeration e = _grammarTable.elements();
        while (e.hasMoreElements()) {
          Object o = e.nextElement();
          if (o instanceof ExternalGrammar) {
            ExternalGrammar g = (ExternalGrammar)o;
            g.resolve(ip);
          }
        }
      }
    */
      public String toString() {
        return getActiveGrammarNames();
      }
    }
```

```
package com.genmagic.invui.vxml.prompt.generator.alphanum;
import com.genmagic.invui.vxml.prompt.CompositePrompt;
import com.genmagic.invui.vxml.prompt.PromptGenerator;
import com.genmagic.invui.vxml.prompt.InvalidFormatException;
public class Handler implements PromptGenerator {
   private String end_qualifier = "fin";
   public Handler() {
   }
   public Handler(String eq) {
      end_qualifier = eq;
   }
   public void appendPrompt(CompositePrompt set, String content) throws InvalidFormatException {
      content = content.trim();
      int len = content.length();
      int end = len-1;
      String wav = "_mid.wav";
      for (int i=0; i<len; i++) {
         char c = content.charAt(i);
         if (Character.isWhitespace(c)) {
            continue;
         }
         if (i == end) {
            wav = "_" + end_qualifier + ".wav";
         }
         if ('0' < c && c < '9') { // Character.isDigit(c)
            set.appendAudio(c + wav,
                     String.valueOf(c));
         } else if ('a' <= c && c <= 'z' ||
               'A' <= c && c <= 'Z') {
            set.appendAudio(Character.toLowerCase(c) + wav,
                     String.valueOf(c));
         } else {
            throw new InvalidFormatException("character is not alpha/num :" + c);
         }
      }
   }
}
```

```
package com.genmagic.invui.vxml.object.method.setSearchPath;
import com.genmagic.invui.vxml.dom.Param;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLException;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.ObjectHandler;
public class Handler implements ObjectHandler {
  public Object handle(VXMLInterp ip,
                       String classid,
                       String data,
                       String type,
                       Param[] params) throws VXMLException {
    if (params == null || params.length != 1) {
      throw new EventException("error.semantic",
                               "only one variable is allowed in submit");
    }
    String value = params[0].getValue();
    if (value == null) {
      Expr expr = params[0].getExpr();
      if (expr != null) {
        value = ip.evalExpression(expr).toString();
      }
    }
    if (value == null) {
      throw new EventException("error.semantic",
                               "no value in parameter");
    }
    ip.getSpeechControl().setSearchPath(value);
    return Boolean.TRUE;
  }
}
```

```
package com.genmagic.invui.vxml.object.method.acceptCall;
import com.genmagic.invui.vxml.dom.Param;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLException;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.ObjectHandler;
public class Handler implements ObjectHandler {
   public Object handle(VXMLInterp ip,
                        String classid,
                        String data,
                        String type,
                        Param[] params) throws VXMLException {
      if (params != null && params.length != 0) {
         throw new EventException("error.semantic",
                              "acceptCall does not take argument");
      }
      ip.getSpeechControl().acceptCall();
      return Boolean.TRUE;
   }
}
```

Docket No. 1-15-0

```
package com.genmagic.invui.vxml.object.method;
import java.util.Hashtable;
import com.genmagic.invui.vxml.dom.Param;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLException;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.ObjectHandler;
public class Handler implements ObjectHandler {
   static private Hashtable _handlers = new Hashtable();
   public Object handle(VXMLInterp ip,
                        String classid,
                        String data,
                        String type,
                        Param[] params) throws VXMLException {
      if (classid.startsWith("method://") == false) {
         throw new EventException("error.semantic", "wrong format :" + classid);
      }
      String method_name = classid.substring(9);
      ObjectHandler handler = (ObjectHandler)_handlers.get(method_name);
      if (handler == null) {
         synchronized(Handler.class) {
            handler = (ObjectHandler)_handlers.get(method_name);
            if (handler == null) {
               handler = createHandler(method_name);
               _handlers.put(method_name, handler);
            }
         }
      }
      return handler.handle(ip, classid, data, type, params);
   }
   static private ObjectHandler createHandler(String method_name) throws EventException {
      String class_name =
         "com.genmagic.invui.vxml.object.method." + method_name + ".Handler";
      try {
         return (ObjectHandler) Class.forName(class_name).newInstance();
      } catch (ClassCastException ex) {
         throw new EventException("error.semantic",
                                  "handler not found :" +
                                  method_name);
      } catch (ClassNotFoundException ex) {
         throw new EventException("error.semantic",
                                  "handler not found :" +
                                  method_name);
      } catch (InstantiationException ex) {
         throw new EventException("error.semantic",
                                  "cannot instantiate object handler :"+
                                  method_name);
      } catch (IllegalAccessException ex) {
         throw new EventException("error.semantic",
                                  "cannot instantiate object handler :"+
                                  method_name);
      }
   }
}
```

```
package com.genmagic.invui.vxml.object.http;
import java.net.URL;
import java.io.IOException;
import java.io.Reader;
import java.io.InputStreamReader;
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import org.w3c.dom.DOMException;
import org.xml.sax.SAXException;
import com.genmagic.invui.vxml.dom.Param;
import com.genmagic.invui.vxml.dom.Expr;
//import com.genmagic.invui.vxml.expr.ExprConstants;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.XMLLoader;
import com.genmagic.invui.vxml.interp.VXMLConstants;
import com.genmagic.invui.vxml.interp.VXMLException;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.GotoNextItemException;
import com.genmagic.invui.vxml.interp.ObjectHandler;
public class Handler implements ObjectHandler {
   public Object handle(VXMLInterp ip,
                        String classid,
                        String data,
                        String type,
                        Param[] params) throws VXMLException {
      XMLLoader loader = ip.getAppContext().getLoader();
      try {
         loader = (XMLLoader)loader.createLoader(classid);
         if (params != null && params.length>0) {
            for (int i=0; i<params.length; i++) {
               Param param = params[i];
               String varname = param.getName();
               String value = param.getValue();
               if (value == null) {
                  Expr expr = param.getExpr();
                  if (expr != null) {
                     value = ip.evalExpression(expr).toString();
                  }
               }
               if (value == null || ip.isUndefined(value)) {
                  throw new EventException("error.semantic",
                                 "no value in parameter");
               }
/*
               java.lang.Object value = ip.getVariable(varname);
               if (value == null || value == ExprConstants.UNDEFINED_OBJ) {
                  throw new EventException("error.semantic",
                                 "variable not defined :" +
                                 varname);
               }
*/
               loader.addParameter(varname, value);
            }
         }
         ip.info("object tag base = " + loader.getDocumentBase() +
```

```
                ", file = " + loader.getFile());
            loader.loadDocument(VXMLConstants.GET);
            Element root = loader.getDocument().getDocumentElement();
            for (Node child = root.getFirstChild();
                child != null;
                child = child.getNextSibling()) {
               ip.eval(child);
            }
         } catch (IOException ex) {
            throw new EventException("error.object.io",
                                     "IO Error occured " +
                                     ex.getMessage(),
                                     ex);
         } catch (SAXException ex) {
            throw new EventException("error.object.sax",
                                     "SAX Error occured " +
                                     ex.getMessage(),
                                     ex);
         } catch (DOMException ex) {
            throw new EventException("error.object.dom",
                                     "DOM Exception occured " +
                                     ex.getMessage(),
                                     ex);
         }
         return Boolean.TRUE;
      }
   }
```

```
package com.genmagic.invui.vxml.prompt.generator.phone;
import java.util.Vector;
import com.genmagic.invui.vxml.prompt.InvalidFormatException;
import com.genmagic.invui.vxml.prompt.CompositePrompt;
import com.genmagic.invui.vxml.prompt.PromptGenerator;
import com.genmagic.invui.vxml.prompt.generator.digits.DigitsPrompt;
public class Handler implements PromptGenerator {
   private String end_qualifier = "fin";
   public Handler() {
   }
   public Handler(String eq) {
      end_qualifier = eq;
   }
   public void appendPrompt(CompositePrompt set, String content) throws InvalidFormatException {
      content = content.trim();
      StringBuffer b = new StringBuffer(10);
      for (int i=0; i< content.length(); i++) {
         char c = content.charAt(i);
         if (Character.isWhitespace(c) ||
            c == '-') {
            continue;
         }
         if (Character.isDigit(c) == false) {
            throw new InvalidFormatException("error.semantic");
         }
         b.append(c);
      }
      content = b.toString();
      if (content.length() != 10) {
         throw new InvalidFormatException("phone number should be 10 digits");
      }
      String area = content.substring(0,3);
      String local = content.substring(3,6);
      String high = content.substring(6,8);
      String low = content.substring(8,10);
      DigitsPrompt.appendDigitsPrompt(set, area, "fin");
      set.appendSilence(200);
      DigitsPrompt.appendDigitsPrompt(set, local, "fin");
      set.appendSilence(200);
      DigitsPrompt.appendDigitsPrompt(set, high, "mid");
      DigitsPrompt.appendDigitsPrompt(set, low, end_qualifier);
   }
}
```

```
package com.genmagic.invui.vxml.prompt.generator.number;
import com.genmagic.invui.vxml.prompt.InvalidFormatException;
import com.genmagic.invui.vxml.prompt.CompositePrompt;
import com.genmagic.invui.vxml.prompt.PromptGenerator;
public class Handler implements PromptGenerator {
   static private final PromptGenerator digitHandlerFin = new
com.genmagic.invui.vxml.prompt.generator.digits.Handler("fin");
   static private final PromptGenerator digitHandlerMid = new
com.genmagic.invui.vxml.prompt.generator.digits.Handler("mid");
   private PromptGenerator digitHandler = digitHandlerFin;
   private String end_qualifier = "fin";
   public Handler() {
   }
   public Handler(String eq) {
      end_qualifier = eq;
      if ("mid".equals(eq)) {
         digitHandler = digitHandlerMid;
      }
   }

   public void appendPrompt(CompositePrompt set, String content) throws InvalidFormatException {
      String number = content.trim();
      String digit = null;
      if (number.indexOf('-') == 0) {
         set.appendAudio("minus_mid.wav", "minus");
         number = number.substring(1);
      }
      int decimal = number.indexOf('.');  // decimal point
      if (decimal>=0) {
         digit = number.substring(decimal+1).trim();
         number = number.substring(0, decimal).trim();
      }
      if (number.length()>0) {
         long l = Long.parseLong(number);
         if (digit != null && digit.length()>0) {
            NumberPrompt.appendPrompt(set, l, "mid");
         } else {
            if (l == 0) {
               set.appendAudio("0_fin.wav", "zero");
            } else {
               NumberPrompt.appendPrompt(set, l, end_qualifier);
            }
         }
      }
      if (digit != null && digit.length()>0) {
         set.appendAudio("point.wav", "point");
         digitHandler.appendPrompt(set, digit);
      }
   }
}
```

```
package com.genmagic.invui.vxml.prompt.generator.duration;
import com.genmagic.invui.vxml.prompt.InvalidFormatException;
import com.genmagic.invui.vxml.prompt.CompositePrompt;
import com.genmagic.invui.vxml.prompt.PromptGenerator;
import com.genmagic.invui.vxml.lex.Lex;
import com.genmagic.invui.vxml.lex.Token;
import com.genmagic.invui.vxml.lex.TokenReader;
import com.genmagic.invui.vxml.lex.LexConstants;
import com.genmagic.invui.vxml.lex.LexException;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.prompt.generator.number.NumberPrompt;
public class Handler implements PromptGenerator, LexConstants {
  String end_qualifier = "_fin.wav";
  public Handler() {
  }
  public Handler(String eq) {
    end_qualifier = "_" + eq + ".wav";
  }
  public void appendPrompt(CompositePrompt set, String content) throws InvalidFormatException {
    try {
      Lex lex = new Lex(content);
      int[] time = new int[5];
      int index = 0;
      int len = 0;
      Token token = lex.next();
      if (token.ttype != INTEGER) {
        throw new InvalidFormatException("type = " + token.ttype);
      }
      time[index++] = (int)token.ltoken;
      token = lex.next();
      while ( token.ttype == CHOICE2) {
        if (time.length == index) {
          throw new InvalidFormatException("type = " + token.ttype);
        }
        token = lex.next();
        if (token.ttype != INTEGER) {
          throw new InvalidFormatException("type = " + token.ttype);
        }
        time[index++] = (int)token.ltoken;
        token = lex.next();
      }
      if (token.ttype != EOE) {
        throw new InvalidFormatException("type = " + token.ttype);
      }
      if (index != 5) {
        int[] tmp = new int[index];
        System.arraycopy(time, 0, tmp, 0, index);
        time = tmp;
      }
      appendPrompts(set, time);

    } catch (LexException ex) {
      throw new InvalidFormatException("lex error :" + ex.getMessage());
    }
  }
  String singular[] = {"year", "day", "hour", "minute", "second"};
```

```
String plural[] = {"years", "days", "hours", "minutes", "seconds"};
public void appendPrompts(CompositePrompt set, int[] time) {
    int index = 0;
    String ap = null;
    int length = time.length;
    for (; length >0 && time[length-1] == 0 ; length--);
    int j = 5 - time.length;
    for (; j<5; j++) {
        int i = time[index++];
        if (i == 0) {
            continue;
        }
        if (ap != null) {
            if (index == length) {
                set.appendAudio(ap + "_and.wav", ap);
            } else {
                set.appendAudio(ap + "_mid.wav", ap);
            }
        }
        if (i == 1) {
            set.appendAudio("1_mid.wav", "1");
            ap = singular[j];
        } else if (i != 0) {
            NumberPrompt.appendPrompt(set, i, "mid");
            ap = plural[j];
        }
    }
    if (ap != null) {
        set.appendAudio(ap + end_qualifier, ap);
    }
  }
}
```

```
package com.genmagic.invui.vxml.prompt.generator.dtmf;
import com.genmagic.invui.vxml.prompt.InvalidFormatException;
import com.genmagic.invui.vxml.prompt.CompositePrompt;
import com.genmagic.invui.vxml.prompt.PromptGenerator;
public class Handler implements PromptGenerator {
   private static final String dtmfChars = "123456789*#";
   public void appendPrompt(CompositePrompt set, String content) throws InvalidFormatException {
      StringBuffer b = new StringBuffer();
      int len = content.length();
      for (int i=0; i<len; i++) {
         char c = content.charAt(i);
         if (Character.isWhitespace(c)) {
            continue;
         } else if (dtmfChars.indexOf(c) >= 0) {
            b.append(c);
         } else {
            throw new InvalidFormatException("invalid character in dtmf :" +
                                    content);
         }
      }
      set.appendDTMF(b.toString());
   }
}
```

Document Page: 2

```
package com.genmagic.invui.vxml.prompt.generator.dollar;
import java.text.*;
import com.genmagic.invui.vxml.prompt.InvalidFormatException;
import com.genmagic.invui.vxml.prompt.CompositePrompt;
import com.genmagic.invui.vxml.prompt.PromptGenerator;
import com.genmagic.invui.vxml.prompt.generator.number.NumberPrompt;
public class Handler implements PromptGenerator {
   private String end_qualifier = "fin";
   private String zero_dollar_fin = "dollar_fin.wav";
   private String cent_fin = "cent_fin.wav";
   private String cents_fin = "cents_fin.wav";
   public Handler() {
   }
   public Handler(String eq) {
      end_qualifier = eq;
      zero_dollar_fin = "dollar_" + eq + ".wav";
      cent_fin = "cent_" + eq + ".wav";
      cents_fin = "cents_" + eq + ".wav";
   }
   public void appendPrompt(CompositePrompt set, String content) throws InvalidFormatException {
      content = content.trim();
      if (content.indexOf('-') == 0) {
         set.appendAudio("minus_mid.wav", "minus");
         content = content.substring(1);
      }
      NumberFormat nf = NumberFormat.getInstance();
      nf.setParseIntegerOnly(true);
      ParsePosition pp = new ParsePosition(0);
      Number dollarNumber = nf.parse(content, pp);
      int len = content.length();
      long dollar = dollarNumber == null ? 0 : dollarNumber.longValue();
      if (len == pp.getIndex()) {
         appendPrompt(set, dollar, 0);
         return;
      }
      if (content.charAt(pp.getIndex()) != '.') {
         throw new InvalidFormatException("format error :" + content);
      }
      // skip decimal point
      String centsStr = content.substring(pp.getIndex()+1).trim();
      len = centsStr.length();
      pp.setIndex(0);
      if (len>2) {
         centsStr = centsStr.substring(0,2);
         len = 2;
         // throw new InvalidFormatException("format error :" + content);
      }
      Number centsNumber = nf.parse(centsStr, pp);
      long cents = centsNumber == null ? 0 : centsNumber.longValue();
      if (len == 1) {
         // 0.1 is 10 cents
         cents *= 10;
      }
      if (len != pp.getIndex()) {
         // or should we accept "$", "dollars", etc?
         throw new InvalidFormatException("format error :" + content);
```

```
    }
    appendPrompt(set, dollar, cents);
  }
  public void appendPrompt(CompositePrompt set,
                           long dollar,
                           long cents) {
    if (dollar> 0) {
      NumberPrompt.appendPrompt(set, dollar, "mid");
      String qual = (cents == 0 ? end_qualifier : "and");
      if (dollar == 1 || dollar == 0) {
        set.appendAudio("dollar_" + qual + ".wav", "dollar");
      } else {
        set.appendAudio("dollars_" + qual + ".wav", "dollars");
      }
    } else if (dollar == 0 && cents == 0) {
      set.appendAudio("0_mid.wav", "zero");
      set.appendAudio(zero_dollar_fin, "dollar");
    }
    if (cents>0) {
      NumberPrompt.appendPrompt(set, cents, "mid");
      if (cents == 1) {
        set.appendAudio(cent_fin, "cent");
      } else {
        set.appendAudio(cents_fin, "cents");
      }
    }
  }
}
```

```
package com.genmagic.invui.vxml.prompt.generator.digits;
import com.genmagic.invui.vxml.prompt.InvalidFormatException;
import com.genmagic.invui.vxml.prompt.CompositePrompt;
import com.genmagic.invui.vxml.prompt.PromptGenerator;
import java.util.Stack;
public class Handler implements PromptGenerator {
   private String end_qualifier = "fin";
   public Handler() {
   }
   public Handler(String eq) {
      end_qualifier = eq;
   }
/*
   public void appendPrompt2(CompositePrompt set, String content) throws InvalidFormatException {
      content = content.trim();
      int len = content.length();
      String qual = "mid";
      for (int i=0 ; i<len; i++) {
         char c = content.charAt(i);
         if (Character.isWhitespace(c)) {
            continue;
         }
         if (Character.isDigit(c) == false) {
            throw new InvalidFormatException("not digit");
         }
         if (len == (i+1)) {
            qual = "fin";
         }
         DigitsPrompt.appendDigitsPrompt(set, String.valueOf(c), qual);
      }
   }
*/
   public void appendPrompt(CompositePrompt set, String content) throws InvalidFormatException {
      StringBuffer b = new StringBuffer(" ");
      int ii = 1;
      content = content.trim();
      Stack stack = new Stack();
      for (int i=content.length()-1; i>=0; i--) {
         char c = content.charAt(i);
         if (Character.isWhitespace(c)) {
            continue;
         }
         if (Character.isDigit(c) == false) {
            throw new InvalidFormatException("not digit");
         }
         b.setCharAt(ii--, c);
         if (ii <0) {
            stack.push(b.toString());
            ii = 1;
            b = new StringBuffer(" ");
         }
      }
      content = b.toString().trim();
      if (content.length()>0) {
         stack.push(content);
      }
```

```
        while(stack.empty()==false) {
            String d = (String) stack.pop();
            String qual = stack.empty() ? end_qualifier : "mid";
            DigitsPrompt.appendDigitsPrompt(set, d, qual);
        }
    }
}
```

Oracle Exhibit 2050

```
package com.genmagic.invui.vxml.prompt.generator.date;
import java.util.Hashtable;
import com.genmagic.invui.vxml.prompt.InvalidFormatException;
import com.genmagic.invui.vxml.prompt.CompositePrompt;
import com.genmagic.invui.vxml.prompt.PromptGenerator;
import com.genmagic.invui.vxml.prompt.generator.number.NumberPrompt;
import com.genmagic.invui.vxml.lex.Lex;
import com.genmagic.invui.vxml.lex.Token;
import com.genmagic.invui.vxml.lex.TokenReader;
import com.genmagic.invui.vxml.lex.LexConstants;
import com.genmagic.invui.vxml.lex.LexException;
import com.genmagic.invui.vxml.interp.EventException;
public class Handler implements PromptGenerator, LexConstants {
   static String[] month = {
      "january.wav",
      "february.wav",
      "march.wav",
      "april.wav",
      "may.wav",
      "june.wav",
      "july.wav",
      "august.wav",
      "september.wav",
      "october.wav",
      "november.wav",
      "december.wav"
   };
   static String[] monthTTS = {
      "january",
      "february",
      "march",
      "april",
      "may",
      "june",
      "july",
      "august",
      "september",
      "october",
      "november",
      "december",
   };
   static String[] monthABR = {
      "jan.",
      "feb.",
      "mar.",
      "apr.",
      "may.",
      "jun.",
      "jul.",
      "aug.",
      "sep.",
      "oct.",
      "nov.",
      "dec.",
   };
   static Hashtable _dayOfWeek = new Hashtable();
```

```
static Hashtable _dayOfWeekTTS = new Hashtable();
static Hashtable _special_mid = new Hashtable();
static Hashtable _special_fin = new Hashtable();
static {
   _dayOfWeek.put("mon",      "monday.wav");
   _dayOfWeek.put("monday",   "monday.wav");
   _dayOfWeek.put("tue",      "tuesday.wav");
   _dayOfWeek.put("tuesday",  "tuesday.wav");
   _dayOfWeek.put("wed",      "wednesday.wav");
   _dayOfWeek.put("wednesday", "wednesday.wav");
   _dayOfWeek.put("thu",      "thursday.wav");
   _dayOfWeek.put("thur",     "thursday.wav");
   _dayOfWeek.put("thursday", "thursday.wav");
   _dayOfWeek.put("fri",      "friday.wav");
   _dayOfWeek.put("friday",   "friday.wav");
   _dayOfWeek.put("sat",      "saturday.wav");
   _dayOfWeek.put("saturday", "saturday.wav");
   _dayOfWeek.put("sun",      "sunday.wav");
   _dayOfWeek.put("sunday",   "sunday.wav");
   _dayOfWeekTTS.put("mon",      "monday");
   _dayOfWeekTTS.put("monday",   "monday");
   _dayOfWeekTTS.put("tue",      "tuesday");
   _dayOfWeekTTS.put("tuesday",  "tuesday");
   _dayOfWeekTTS.put("wed",      "wednesday");
   _dayOfWeekTTS.put("wednesday", "wednesday");
   _dayOfWeekTTS.put("thu",      "thursday");
   _dayOfWeekTTS.put("thur",     "thursday");
   _dayOfWeekTTS.put("thursday", "thursday");
   _dayOfWeekTTS.put("fri",      "friday");
   _dayOfWeekTTS.put("friday",   "friday");
   _dayOfWeekTTS.put("sat",      "saturday");
   _dayOfWeekTTS.put("saturday", "saturday");
   _dayOfWeekTTS.put("sun",      "sunday");
   _dayOfWeekTTS.put("sunday",   "sunday");
   _special_fin.put("today",    "today.wav");
   _special_fin.put("tomorrow", "tomorrow.wav");
   _special_fin.put("yesterday", "yesterday.wav");
   _special_mid.put("today",    "today_mid.wav");
   _special_mid.put("tomorrow", "tomorrow_mid.wav");
   _special_mid.put("yesterday", "yesterday_mid.wav");
}
private String end_qualifier = "fin";
private Hashtable _special = _special_fin;
public Handler() {
}
public Handler(String eq) {
   end_qualifier = eq;
   if ("mid".equals(eq)) {
      _special = _special_mid;
   }
}
public void appendPrompt(CompositePrompt set, String content) throws InvalidFormatException {
   try {
      Lex lex = new Lex(content);
      Token token = lex.next();
      if (token.ttype == VARIABLE ||
```

```
    token.ttype == STRING) {
    String str = token.stoken.toLowerCase();
    Object obj =_special.get(str);
    if (obj != null) {
      // today/tomorrow/yesterday
      set.appendAudio((String)obj, str);
      token = lex.next();
      if (token.ttype == EOE) {
        return;
      }
      throw new InvalidFormatException("type = " + token.ttype);
    }
    String wav = (String)_dayOfWeek.get(str);
    if (wav == null) {
      throw new InvalidFormatException("unknown day of week");
    }
    String tts = (String)_dayOfWeekTTS.get(str);
    set.appendAudio(wav, tts);
    token = lex.next();
  }
  int first = 0;
  if (token.ttype == INTEGER) {
    first = (int)token.ltoken;
  } else if (token.ttype == VARIABLE ||
          token.ttype == STRING) {
    for (int i=0; i<monthTTS.length; i++) {
      if (monthTTS[i].equalsIgnoreCase(token.stoken)) {
        first = i+1;
        break;
      } else if (monthABR[i].equalsIgnoreCase(token.stoken)) {
        first = i+1;
        break;
      }
    }
  }
  if (first == 0) {
    throw new InvalidFormatException("type = " + token.ttype);
  }
  token = lex.next();
  if (token.ttype == MULOP && token.ctoken == '/') {
    token = lex.next();
  }
  if (token.ttype != INTEGER) {
    throw new InvalidFormatException("type = " + token.ttype);
  }
  int second = (int)token.ltoken;
  token = lex.next();
  if (token.ttype == EOE) {
    appendPrompts(-1, first, second, set);
    return;
  }
  if (token.ttype != MULOP || token.ctoken != '/') {
    throw new InvalidFormatException("type = " + token.ttype);
  }
  token = lex.next();
  if (token.ttype != INTEGER) {
```

```
            throw new InvalidFormatException("type = " + token.ttype);
        }
        int third = (int)token.ltoken;
        token = lex.next();
        if (token.ttype != EOE) {
            throw new InvalidFormatException("type = " + token.ttype);
        }
        appendPrompts(first, second, third, set);
        return;
    } catch (LexException ex) {
        throw new InvalidFormatException("lex error :" +
                                ex.getMessage());
    }
}
private void appendPrompts(int year, int month, int date, CompositePrompt set) throws InvalidFormatException {

    if (year == -1) {
        // month and date
        String monthWav = toAudioMonth(month - 1);
        String dateWav = toAudioOrdinal(date, end_qualifier);

        set.appendAudio(monthWav, monthTTS[month-1]);
        set.appendAudio(dateWav,  getDateTTS(date));
    } else if (999 < year && year < 10000) {
        // year, month and date
        String monthWav = toAudioMonth(month - 1);
        String dateWav = toAudioOrdinal(date, "mid");
        set.appendAudio(monthWav, monthTTS[month-1]);
        set.appendAudio(dateWav,  getDateTTS(date));
        appendYear(set, year);
    } else {
        throw new InvalidFormatException("year = " + year);
    }
}
private void appendYear(CompositePrompt set, int year) {
    if ( (1900 <= year && year <= 1909) ||
         (2000 <= year && year <= 2009) ||
         (2100 <= year && year <= 2109)) {
        set.appendAudio("year_" + year + ".wav",
                    String.valueOf(year));
    } else {
        String yearStr = String.valueOf(year);
        String high = yearStr.substring(0, 2);
        String low = yearStr.substring(2);
        set.appendAudio(high + "_mid.wav", high);
        set.appendAudio(low + "_" + end_qualifier + ".wav", low);
    }
}
static private String getDateTTS(int date) {
    switch(date) {
    case 1:
    case 21:
    case 31:
        return date + "st";
    case 2:
    case 22:
```

```
            return date + "nd";
         case 3:
         case 23:
            return date + "rd";
         default:
            return date + "th";
         }
      }
      /**
       * Converters
       */
      static private String toAudioHours(int i) {
         return "time_hour_" + i + ".wav";
      }
      static private String toAudioMinutes(int i) {
         return "time_min_" + i + ".wav";
      }
      static private String toAudioNumber(int i, String type) {
         return i + "_" + type + ".wav";
      }
      static private String toAudioMonth(int i) {
         return month[i];
      }
      static private String toAudioOrdinal(int i, String type) {
         return "ord_" + i + "_" + type + ".wav";
      }
   }
```

```
package com.genmagic.invui.vxml.prompt.generator.currency;
import com.genmagic.invui.vxml.prompt.InvalidFormatException;
import com.genmagic.invui.vxml.prompt.CompositePrompt;
import com.genmagic.invui.vxml.prompt.PromptGenerator;
import com.genmagic.invui.vxml.prompt.PromptGeneratorFactory;
public class Handler implements PromptGenerator {
   static final private PromptGenerator _dollar = new com.genmagic.invui.vxml.prompt.generator.dollar.Handler();
   public Handler() {
   }
   public void appendPrompt(CompositePrompt set, String content) throws InvalidFormatException {
      content = content.trim();
      if (content.length() >0) {
         if (content.charAt(0) == '$') {
            content = content.substring(1);
         } else if (content.charAt(content.length()-1) == '$') {
            content = content.substring(0, content.length() - 1);
         } else {
            throw new InvalidFormatException("Unknown Currency " + content);
         }
         _dollar.appendPrompt(set, content);
      }
      throw new InvalidFormatException("Unknown Currency " + content);
   }
}
```

```
package com.genmagic.invui.vxml.prompt.generator.time;
import com.genmagic.invui.vxml.prompt.InvalidFormatException;
import com.genmagic.invui.vxml.prompt.CompositePrompt;
import com.genmagic.invui.vxml.prompt.PromptGenerator;
import com.genmagic.invui.vxml.lex.Lex;
import com.genmagic.invui.vxml.lex.Token;
import com.genmagic.invui.vxml.lex.TokenReader;
import com.genmagic.invui.vxml.lex.LexConstants;
import com.genmagic.invui.vxml.lex.LexException;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.prompt.generator.number.NumberPrompt;
public class Handler implements PromptGenerator, LexConstants {
   private String noon_wav = "noon.wav";
   private String midnight_wav = "midnight.wav";
   private String am_wav = "am.wav";
   private String pm_wav = "pm.wav";
   public Handler() {
   }
   public Handler(String eq) {
      noon_wav = "noon_" + eq + ".wav";
      midnight_wav = "midnight_" + eq + ".wav";
      am_wav = "am_" + eq + ".wav";
      pm_wav = "pm_" + eq + ".wav";
   }
   public void appendPrompt(CompositePrompt set, String content) throws InvalidFormatException {
      try {
         Lex lex = new Lex(content);
         Token token = lex.next();
         if (token.ttype != INTEGER) {
            throw new InvalidFormatException("type = " + token.ttype);
         }
         int first = (int)token.ltoken;
         token = lex.next();
         if (token.ttype != CHOICE2) {
            throw new InvalidFormatException("type = " + token.ttype);
         }
         token = lex.next();
         if (token.ttype != INTEGER) {
            throw new InvalidFormatException("type = " + token.ttype);
         }
         int second = (int)token.ltoken;
         token = lex.next();
         boolean am = true;
         if (token.ttype == EOE) {
            if (first > 11) {
               first -= 12;
               am = false;
            }
            appendPrompts(set, first, second, am);
         } else if (token.ttype == VARIABLE ||
               token.ttype == STRING) {
            if ("am".equalsIgnoreCase(token.stoken)) {
               am = true;
            } else if ("pm".equalsIgnoreCase(token.stoken)) {
               am = false;
            }
```

```
            appendPrompts(set, first, second, am);
          } else {
            throw new InvalidFormatException("type = " + token.ttype);

          }
        } catch (LexException ex) {
          throw new InvalidFormatException("lex error :" + ex.getMessage());
        }
      }
      public void appendPrompts(CompositePrompt set,
                        int hours, int minutes, boolean am) {
        if (hours == 12 && minutes == 0) {
          set.appendAudio(toAudioMinutes(hours),
                    String.valueOf(hours));
          if (am) {
            set.appendAudio(midnight_wav, "midnight");
          } else {
            set.appendAudio(noon_wav, "noon");
          }

        } else {
          set.appendAudio(toAudioHours(hours),
                    String.valueOf(hours));
/*
          if (minutes < 10) {
            set.appendAudio("o_mid.wav", "oh");
          }
*/
          if (minutes != 0) {
            set.appendAudio(toAudioMinutes(minutes),
                      String.valueOf(minutes));
          }
          if (am) {
            set.appendAudio(am_wav, "am");
          } else {
            set.appendAudio(pm_wav, "pm");
          }
        }
      }
      /**
       * Converters
       */
      static private String toAudioHours(int i) {
        return "time_hour_" + i + ".wav";
      }
      static private String toAudioMinutes(int i) {
        return "time_min_" + i + ".wav";
      }
    }
```

```
package com.genmagic.invui.vxml.dom;
import org.w3c.dom.Node;
import com.genmagic.invui.vxml.interp.EventException;
abstract public class If extends VXMLElement {
   // TBD
   public If() {
      super("if");
   }
   static public interface IfBlock {
      public Expr getExpr() throws EventException;
      public Node[] getBlock();
   }
   abstract public IfBlock[] getIfBlocks();
}
```

```
package com.genmagic.invui.vxml.tags;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.dom.If;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.interp.TagHandler;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLException;
import com.genmagic.invui.vxml.interp.EventException;
public class IfTag implements TagHandler {
   public void perform(VXMLInterp ip, Element self) throws VXMLException {
      If _if = (If)self;
      If.IfBlock[] blocks = _if.getIfBlocks();
      for (int i=0; i<blocks.length ; i++) {
         If.IfBlock b = blocks[i];
         Expr expr = b.getExpr();
         if (expr == null ||
            ip.evalBooleanExpression(expr)) {
            ip.eval(b.getBlock());
            return;
         }
      }
   }
}
```

```
package com.genmagic.invui.vxml.dom;
import org.w3c.dom.Node;
import com.genmagic.invui.vxml.interp.GrammarSet;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.VXMLInterp;
abstract public class Initial extends FormItem {
   public Initial() {
      super("initial");
   }
   /**
    * Returns prompt related children only.
    * prompt, audio, PCDATA, value, enumerate
    */
   abstract public Node[] getPromptElements();
   /**
    * @deprecated as of 1.0
    */
   abstract public int getTimeout();
   /**
    * @deprecated as of 1.0
    */
   abstract public int getToomuchTimeout();
   /**
    * @deprecated as of 1.0
    */
   abstract public int getEndspeechTimeout();
   /**
    * Get active grammar definitions on this form item.
    */
   abstract public GrammarSet getActiveGrammars(VXMLInterp ip) throws EventException;
   /**
    * Get Grammars (since initial may have link)
    */
   abstract public com.genmagic.invui.vxml.interp.Grammar[] getGrammars(VXMLInterp ip) throws EventException;
}
```

```
package com.genmagic.invui.vxml.tags;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.dom.Initial;
import com.genmagic.invui.vxml.dom.Catch;
import com.genmagic.invui.vxml.dom.Filled;
import com.genmagic.invui.vxml.interp.*;
public class InitialTag implements TagHandler, VXMLConstants {
  public void perform(VXMLInterp ip, Element self) throws VXMLException {
    Initial initial = (Initial)self;
    SpeechControl speech = ip.getSpeechControl();
    GrammarSet grammarSet = initial.getActiveGrammars(ip);
    XMLLoader loader = ip.getAppContext().getLoader();
    String stateName =
      loader.getDocumentBase() +
      loader.getFile() + "#" +
      initial.getStateName();
    // speech.setReprompt(true);
    // speech.backupPrompts();
    String varName = initial.getName();
    Properties props = ip.getCurrentProperties();
    try {
      speech.enterState(stateName);
      if (speech.getReprompt()) {
        ip.eval(initial.getPromptElements());
        speech.setReprompt(false);
      }
      // get property after we evaluated prompts
      int timeout = speech.getTimeout();
      if (timeout<0) {
        timeout = (int)props.getTimeProperty("timeout",
                                DEFAULT_REC_TIMEOUT);
      }
      speech.clearTimeout();

      int toomuch = (int)props.getTimeProperty("toomuchtimeout",
                                DEFAULT_TOOMUCH_TIMEOUT);

      int endspeech = (int)props.getTimeProperty("endspeechtimeout",
                                  DEFAULT_ENDSPEECH_TIMEOUT);
      RecResult result =
        speech.playAndRecognition(grammarSet,
                        timeout,
                        toomuch,
                        endspeech,
                        DEFAULT_MAX_DIGIT);
      result.performAction(ip);
    } catch (GotoNextItemException ex) {
      ip.setVariable(varName, Boolean.TRUE);
      throw ex;
    } catch(VXMLException ex) {
      throw ex;
    } catch(RuntimeException ex) {
      throw ex;
    } catch (Exception ex) {
      ex.printStackTrace();
```

```
            throw new EventException("error.internal", ex.getMessage());
          } finally {
            speech.leaveState();
          }
          throw new EventException("nomatch");
        }
      }
```

Document ID: 1020020.1

```
package com.genmagic.invui.vxml.prompt;
import com.genmagic.invui.vxml.interp.EventException;
public class InvalidFormatException extends EventException {
   public InvalidFormatException(String exception) {
      super("error.invalidformat", exception);
   }
   public InvalidFormatException(String name, String exception) {
      super(name, exception);
   }
}
```

Oracle - Exhibit 2010

```
package com.genmagic.invui.vxml.dom;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.VXMLInterp;
abstract public class Link extends VXMLElement implements DocumentFetchable {
   public Link() {
      super("link");
   }
   abstract public String getNext();
   /**
    * @since Vxml1.0
    */
   abstract public Expr getExpr() throws EventException;
   abstract public String getEventName();
   abstract public com.genmagic.invui.vxml.interp.Grammar[] getLinkGrammars(VXMLInterp ip) throws
EventException;
}
```

```
package com.genmagic.invui.vxml.domimpl;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.dom.VXMLElement;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.Util;
class LinkInfo {
   VXMLElement e;
   String next = null;
   Expr expr = null;
   LinkInfo(VXMLElement e) {
      this.e = e;
      next = e.getAttribute2("next");
   }
   Expr getExpr() throws EventException {
      if (expr == null) {
         expr = Util.getExprAttribute(e, "expr");
      }
      return expr;
   }
}
```

```
package com.genmagic.invui.vxml.interp;
import java.io.*;
import java.net.URL;
import java.net.URLConnection;
import java.net.MalformedURLException;
public interface Loader {
  /**
   * Sets the path used to load a document. The path can be relative
   * to the document base, or full url.
   * @param fileOrUrl the filename or url of the document.
   */
  public void setPath(String fileOrUrl) throws MalformedURLException;
  /**
   * Adds parameter used in a request.
   */
  public void addParameter(String name, Object value);
  /**
   * Creates new class loader with new path of the document to load.
   * The path may be relative path to the document base of this XMLLoader.
   * @param path a path of the document to load. This can be relative path
   *          to the document base of this loader.
   * @return an XMLLoader which has new path information.
   */
  public Loader createLoader(String path) throws java.net.MalformedURLException;
  public Loader cloneLoader();
  public boolean sameURL(Loader loader);
  /**
   * Open Stream
   * @method
   * @timeout read timeout for input stream.
   */
  public URLConnection openConnection(int method, int timeout, String enctype) throws IOException;
  /**
   * Open Reader
   * @method
   * @timeout read timeout for input stream.
   * @enctype encording type for parameters
   */
  public Reader openReader(int method, int timeout, String enctype) throws IOException;
  /**
   * Gets the current document's path
   * @return the current document's path
   */
  public String getPath();
  /**
   * Gets the base url of the document
   */
  public URL getDocumentBase();
  /**
   * Gets the file part of the document's path.
   */
  public String getFile();
  /**
   * Gets the anchor of specifid path.
   */
  public String getAnchor();
```

```
    /**
     *
     */
    public String getStateString(int depth);
    /*
     * Close loader
     */
    public void close();
}
```

Document Page

```
package com.genmagic.invui.vxml.dialogengine;
import java.io.*;
import java.net.InetAddress;
import com.genmagic.invui.vxml.interp.VXMLInterpImpl;
import com.genmagic.invui.util.Getopts;
import com.genmagic.invui.util.DefaultLogFactory;
public class Main {
   static Getopts.Entry[] opts = {
      Getopts.entry("-incoming",       "jde.script.incoming",
            "file:/D:/temp/incoming.ddl"),
      Getopts.entry("-mtsgroup",       "jde.mts.group"),
      Getopts.entry("-jdegroup",       "jde.jde.group"),
      Getopts.entry("-enable09",       "jde.voicexml09", "false"),
      Getopts.entry("-accept",         "jde.accept",  "inout"),
      Getopts.entry("-help",           "jde.help", "false"),
      Getopts.entry("-log",            "jde.log",  "stdout"),
      Getopts.entry("-log_state_depth", "jde.log.statedepth",
            "2"),
      Getopts.entry("-mtslog",         "mts.log", "false"),
      Getopts.entry("-emulate",        "jde.emulator.file", ""),
      Getopts.entry("-incapacity",     "jde.capacity.incoming",
            "-1"),
      Getopts.entry("-linecapacity",   "jde.capacity.line",
            "272"),
      Getopts.entry("-disable_bargein", "jde.bargein.disable",
            "false"),
      Getopts.entry("-disable_bargein", "jde.bargein.disable",
            "false"),
   };
   public static void main(String a[]) throws Exception {
      // Initialize the ORB.
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(a, null);
      Getopts o = new Getopts(a);
      o.process(opts);
      if (o.checkError() || o.getBoolean("jde.help")) {
         usage(o.getErrorMsg());
         System.exit(1);
      }
      String logName = System.getProperty("jde.log");
      if ("stdout".equals(logName)) {
         DefaultLogFactory.setOutput(new PrintWriter(System.out));
      } else if ("stderr".equals(logName)) {
         DefaultLogFactory.setOutput(new PrintWriter(System.err));
      } else {
         DefaultLogFactory.setOutputFile(logName);
      }
      String incoming = System.getProperty("jde.script.incoming");
      String mts_group = System.getProperty("jde.mts.group").trim();
      String jde_group = System.getProperty("jde.jde.group").trim();
      String emulateFile = o.getString("jde.emulator.file","").trim();
      DialogEngine de = null;
      if (emulateFile.length()>0) {
/*
         String[][] slotdefs =
            SlotDefReader.readSlotDefs(emulateFile);
         de = new DialogEngine(incoming,
```

```
                    slotdefs,
                    product);
*/
    } else {
      de = new DialogEngine(orb,jde_group, mts_group);
    }
    String acceptStr = o.getString("jde.accept");
    if ("in".equals(acceptStr) || "inout".equals(acceptStr)) {
      de.setAcceptIncomingCall(incoming);
    }
    if ("out".equals(acceptStr) || "inout".equals(acceptStr)) {
      de.setAcceptOutgoingCall();
    }
    de.setMaxNumberOfCalls(Integer.getInteger("jde.capacity.line").intValue());
    de.setMTSLog(Boolean.getBoolean("mts.log"));
    // Wait for incoming requests
    orb.run();
  }
  static void usage(String msg) {
    System.out.println("usage dh [-options]\n\nwhere options include:\n" +
                "  -help          print out this message\n" +
                "  -enable09      enable VoiceXML0.9\n" +
                "  -incoming <file>  the script for incoming calls\n" +
                "  -mtsgroup <name>  the name of mts group\n" +
                "  -jdegroup <name>  the name of jde group\n" +
                "  -log   <file>   the log file name\n" +
                "  -mtslog        enables the logging facility of MTS\n" +
                "  -accept <in/out/inout> specify if dh accepts inbound, outbound or both of them\n" +
                "  -emulate <file> specify the emulation file\n" +
                "  -incapcity <number> specify the number of clients allocated for incoming call. Default is 'dynamic
allocation'.\n" +
                "  -emulate <file>    specify the emulation file\n" +
                "  -linecapacity      the max number of call sessions\n" +
                "  -disable_bargein   disable bargein when specified\n" +
                "  -log_state_depth   specify how many path elements of URL to be logged as a state name\n" +
                msg);
  }
}
```

```
package com.genmagic.invui.vxml.dom;
import org.w3c.dom.Node;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.GrammarSet;
abstract public class Menu extends Dialog {
  // TBD
  public Menu() {
    super("menu");
  }
  /**
   * Returns prompt related children only.
   * prompt, audio, PCDATA, value, enumerate
   */
  abstract public Node[] getPromptElements();
  /**
   * Choice elements in this form
   */
  abstract public Choice[] getChoices();
  /**
   * Get declarations
   * @since as of VXML1.1
   */
  abstract public VXMLElement[] getDecls();
  /**
   * dtmf
   */
  abstract public boolean getDtmf();
  /**
   * Grammar
   */
  abstract public void getGrammar(VXMLInterp ip, GrammarSet set) throws EventException;
  abstract public GrammarSet getActiveGrammars(VXMLInterp ip) throws EventException;
}
```

```
package com.genmagic.invui.vxml.tags;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.dom.Menu;
import com.genmagic.invui.vxml.dom.Catch;
import com.genmagic.invui.vxml.interp.*;
public class MenuTag implements TagHandler, VXMLConstants {
   public void perform(VXMLInterp ip, Element self) throws VXMLException {
      Menu menu = (Menu)self;
      int count = 0;
      SpeechControl speech = ip.getSpeechControl();
      GrammarSet grammarSet = menu.getActiveGrammars(ip);
      XMLLoader loader = ip.getAppContext().getLoader();
      String stateName =
         loader.getDocumentBase() +
         loader.getFile() + "#" +
         menu.getID();
      speech.setReprompt(true);
      speech.backupPrompts();
      EventCounter evc = new EventCounter();
      Properties props = ip.getCurrentProperties();
      int toomuch = (int)props.getTimeProperty("toomuchtimeout",
                              DEFAULT_TOOMUCH_TIMEOUT);
      int endspeech = (int)props.getTimeProperty("endspeechtimeout",
                              DEFAULT_ENDSPEECH_TIMEOUT);
      try {
outer:
         while (true) {
            if (speech.getReprompt()) {
               // Note: This may have to be incremented at "reprompt"
               count ++;
               speech.setCount(count);
               ip.eval(menu.getPromptElements());
               speech.setReprompt(false);
            }
            // get timeout after evaluated the prompts
            int timeout = speech.getTimeout();
            if (timeout<0) {
               timeout = (int)props.getTimeProperty("timeout",
                                    DEFAULT_REC_TIMEOUT);
            }
            speech.clearTimeout();
            Object value = null;
            Catch handler = null;
            try {
               speech.enterState(stateName);
               RecResult result =
                  speech.playAndRecognition(grammarSet,
                              timeout,
                              toomuch,
                              endspeech,
                              DEFAULT_MAX_DIGIT);
               result.performAction(ip);

               throw new EventException("error.internal",
                              "should not reach here");
            } catch (EventException ex) {
```

```
                String eventName = ex.getEventName();
                ip.diag("event exception in recognition :"+
                      eventName + ", msg=" + ex.getMessage());
                handler = menu.getEventHandler(ip, ex,
                                    evc.getCount(eventName));
                if (handler == null) {
                  throw ex;
                }
              } catch (RuntimeEventException ex) {
                String eventName = ex.getEventName();
                ip.diag("event exception in recognition :"+
                      eventName + ", msg=" + ex.getMessage());
                handler = menu.getEventHandler(ip, ex,
                                    evc.getCount(eventName));
                if (handler == null) {
                  throw ex;
                }
              } finally {
                speech.leaveState();
              }
inner:
              while(true) {
                try {
                  Catch c = handler;
                  handler = null;
                  ip.eval(c);

                  continue outer;
                } catch (EventException ex) {
                  String eventName = ex.getEventName();
                  handler = menu.getEventHandler(ip, ex,
                                      evc.getCount(eventName));
                  if (handler == null) {
                    throw ex;
                  }
                  continue inner;
                } catch (RuntimeEventException ex) {
                  String eventName = ex.getEventName();
                  handler = menu.getEventHandler(ip, ex,
                                      evc.getCount(eventName));
                  if (handler == null) {
                    throw ex;
                  }
                  continue inner;
                }
                // break outer;
              }
            }
          } catch(VXMLException ex) {
            throw ex;
          } catch(RuntimeException ex) {
            throw ex;
          } catch(Exception ex) {
            ex.printStackTrace();
            throw new EventException("error.internal", ex.getMessage());
          }
```

```
    }
}
```

Document ID: 0.7.20.9

```
package com.genmagic.invui.vxml.dom;
/**
 */
abstract public class Meta extends VXMLElement {
   public Meta() {
      super("meta");
   }
   abstract public String getName();
   abstract public String getContent();
   abstract public String getHttpEquiv();
}
```

Adobe PageMill 3.0

```
package com.genmagic.invui.vxml.interp;
public interface MTSConstants {
  /**
   * State of outboung call
   */
  static public final int CPN_Proceeding = 0;
  static public final int CPN_Busy      = 1;
  static public final int CPN_NoAnswer  = 2;
  static public final int CPN_NoRingback = 3;
  static public final int CPN_NoDialtone = 4;
  static public final int CPN_Error     = 5;
  static public final String CPN_STATUS_STRING[] = {
    "Proceeding",
    "Busy",
    "noAnswer",
    "NoRingBack",
    "NoDialtone",
    "Error",
  };
  /*
   * State of VUIBrokerClient
   */
  static public final int INCOMING = 0;
  static public final int OUTGOING = 1;
  static public final int IDLE = 2;
  static public final String STATE_STRING[] = {
    "INCOMING",
    "OUTGOING",
    "IDLE"
  };
  /**
   * Rec Status
   */
  static public final int RC_SuccessfulSpeech = 0;
  static public final int RC_SuccessfulDTMF = 1;
  static public final int RC_NoSpeech = 2;
  static public final int RC_TooMuchSpeech = 3;
  static public final int RC_CallDropped = 4;
  static public final int RC_Error = 5;
  static public final int RC_TIMEOUT = 6;
  static public final String REC_STATUS_STRING[] = {
    "SuccessfulSpeech",
    "SuccessfulDTMF",
    "NoSpeech",
    "TooMuchSpeech",
    "CallDropped",
    "Error",
    "Timeout"
  };
  /**
   * Dynamic Grammar Status
   */
  static public final int DG_Success = 0;
  static public final int DG_CantOpenDatabase = 1;
  static public final int DG_CantFindDynamicGrammar = 2;
  static public final int DG_CantFindParentGrammar = 3;
```

```
static public final int DG_Failure = 4;
static public final String DG_STATUS_STRING[] = {
   "Success",
   "CantOpenDatabase",
   "CantFindDynamicGrammar",
   "CantFindParentGrammar",
   "Failure",
};
/**
 * Media Status
 */
static public final int MEDIA_OK = 0;
static public final int MEDIA_ERROR = 1;
/*
 * Info. codes for TSP Failure Notifications
 */
static public final String TSP_MSG[] = {
   "TSPF_OK",
   "TSPF_CANTGOOFFHOOK",
   "TSPF_CANTGOONHOOK",
   "TSPF_INVALIDCALLTOKEN",
   "TSPF_NOAVAILABLECHANNEL",
   "TSPF_INVALIDREQUESTTOKEN",
   "TSPF_CANTDIAL",
   "TSPF_CANTCANCEL",
   "TSPF_CANTLISTEN",
   "TSPF_CALLNOTOFFERED",
   "TSPF_CANTANSWER",
   "TSPF_CANTGETTIMESLOT",
   "TSPF_CANTCREATESESSION",
   "TSPF_INSUFFICIENTRESOURCES",
   "TSPF_NOSERVICEPROVIDER",
   "TSPF_SYSTEMFAILURE",
   "TSPF_CALLALREADYDISCONNECTED",
   "TSPF_RESTRICTEDNUMBER",
};
/*
 * Info. codes for MSP Failure Notifications
 */
static public final String MSP_MSG[] = {
   "MSPF_OK",
   "MSPF_NOTIDLE",
   "MSPF_CANTSTOP",
   "MSPF_CANTFINDPROMPT",
   "MSPF_CANTRECORD",
   "MSPF_CANTPLAY",
   "MSPF_INVALIDCALLTOKEN",
   "MSPF_NOTHINGTOSTOP",
   "MSPF_NOTHINGTOPLAY",
   "MSPF_NOSUCHRECORDING",
   "MSPF_CANTHANDOFFRECORDING",
   "MSPF_TTSSYSTEMFAILURE",
   "MSPF_CANTTTS",
   "MSPF_INSUFFICIENTRESOURCES",
   "MSPF_NOSERVICEPROVIDER",
   "MSPF_CANTPLAYPROMPTSEGMENT",
```

Exhibit 1013, p. 2 of 10

```
    "MSPF_COMPOSITEPROMPTFAILURE",
  };
  /*
   * Info. codes for SSP Failure Notifications
   */
  static public final String SSPF_MSG[] = {
    "SSPF_OK",
    "SSPF_INVALIDCALLTOKEN",
    "SSPF_NOTIDLE",
    "SSPF_NOLICENSE",
    "SSPF_CANTRECOGNIZE",
    "SSPF_RECORDCHANNELNOTIDLE",
    "SSPF_CANTRECORD",
    "SSPF_INSUFFICIENTRESOURCES",
    "SSPF_NOSERVICEPROVIDER",
    "SSPF_UNKNOWNGRAMMAR",
    "SSPF_DATABASEERROR",
    "SSPF_SYSTEMFAILURE",
  };
  /**
   * TTS Parameters
   */
}
```

```
package com.genmagic.invui.vxml.script.expr;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
/** interface for N arg function in expressions
 * @see Expression
 */
interface NargFunction extends Function {
   Object perform(VXMLInterp ip, Object args[]) throws EventException;
}
```

Oracle Ex. 2009, p. 2

```
package com.genmagic.invui.vxml.script.expr;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
/** interface for no arg function in expressions
 * @see Expression
 */
interface NoArgFunction extends Function {
   Object perform(VXMLInterp ip) throws EventException;
}
```

Oracle Ex. 2010, Page 1470

```
package com.genmagic.invui.vxml.interp;
import java.util.Hashtable;
import com.genmagic.invui.util.LogFactory;
/**
 * BuiltInGrammars
 */
public class NoneGrammar extends Grammar {
   public NoneGrammar() {
   }
   public String getName() {
      return "none";
   }
   public String getUniqID() {
      return "none";
   }
   public String[] getSlotNames() {
      return new String[0];
   }
   public float getWeight() {
      return -1;
   }
   public SlotResult getSlotResult(RecResult result, String name) throws EventException {
      return result.getRawResult(name);
   }
   public String toString() {
      return "Builtin:none";
   }
}
```

```
package com.genmagic.invui.vxml.interp;
public class NotImplementedException extends RuntimeException {
   public NotImplementedException(String str) {
      super(str);
   }
}
```

Document provided by IPR2015-00326

```
package com.genmagic.invui.vxml.prompt.generator.number;
import com.genmagic.invui.vxml.prompt.CompositePrompt;
import java.util.Vector;
public class NumberPrompt {
   static private final long trillion = 1000000000000L;
   static private final long billion = 1000000000;
   static private final long million = 1000000;
   public static void appendPrompt(CompositePrompt set, long l, String qual) {
      if (l>=trillion) {
         long tmp = l/trillion;
         appendPrompt(set, tmp, "mid");
         l = l % trillion;
         if (l == 0) {
            set.appendAudio("trillion_" + qual + ".wav", "triillion");
            return;
         } else {
            set.appendAudio("trillion_mid.wav", "trillion");
         }
      }
      if (l>=billion) {
         long tmp = l/billion;
         appendPrompt(set, tmp, "mid");
         l = l % billion;
         if (l == 0) {
            set.appendAudio("billion_" + qual + ".wav", "billion");
            return;
         } else {
            set.appendAudio("billion_mid.wav", "billion");
         }
      }
      if (l>=million) {
         long tmp = l/million;
         appendPrompt(set, tmp, "mid");
         l = l % million;
         if (l == 0) {
            set.appendAudio("million_" + qual + ".wav", "million");
            return;
         } else {
            set.appendAudio("million_mid.wav", "million");
         }
      }
      if (l>=1000) {
         long tmp = l/1000;
         appendPrompt(set, tmp, "mid");
         l = l % 1000;
         if (l == 0) {
            set.appendAudio("thousand_" + qual + ".wav", "thousand");
            return;
         } else {
            set.appendAudio("thousand_mid.wav", "thousand");
         }
      }
      if (l>=100) {
         long tmp = l - (l % 100);
         l = l % 100;
         if (l == 0) {
```

```
            set.appendAudio(tmp + "_" + qual + ".wav", "hundred");
            return;
          } else {
            set.appendAudio(tmp + "_mid.wav", "hundred");
          }
        }
        if (l != 0) {
          set.appendAudio(l + "_" + qual + ".wav", String.valueOf(l));
        }
      }
    }
```

Document Page 3 of 3

```
package com.genmagic.invui.vxml.dom;
abstract public class Object extends FormItem implements DocumentFetchable {
   public Object() {
      super("object");
   }
   /**
    * @since Vxml1.0
    */
   abstract public String getClassid();
   /**
    * @since Vxml1.0
    */
   abstract public String getCodebase();
   /**
    * @since Vxml1.0
    */
   abstract public String getCodetype();
   /**
    * @since Vxml1.0
    */
   abstract public String getData();
   /**
    * @since Vxml1.0
    */
   abstract public String getType();
   /**
    * @since Vxml1.0
    */
   abstract public String getArchive();
   /**
    * @since Vxml1.0
    */
   abstract public Param[] getParams();
   /**
    * @deprecated as of Vxml1.0
    */
   abstract public String getSrc();
   /**
    * @deprecated as of Vxml1.0
    */
   abstract public String[] getSubmit();
   /**
    * @deprecated as of Vxml1.0
    */
   abstract public String[] getExpect();
}
```

```
package com.genmagic.invui.vxml.interp;
import com.genmagic.invui.vxml.dom.Param;
/**
 * The ObjectHandler is the interface for all object tags implementation.
 * The system identifies the object handler for each scheme in src attribute.
 *
 */
public interface ObjectHandler {
   /**
    * Handle the request.
    * @param ip the interpreter
    * @param src the url of the object tag to handle
    * @param submit array of input variable names
    * @param exptct array of output variable names
   public void handle(VXMLInterp ip, String src, String[] submit, String[] expect) throws VXMLException;
    */
   /**
    * Handle the request.
    * @param ip the interpreter
    * @param obj object tag
    * @return result of object tag
    */
   public Object handle(VXMLInterp ip,
                        String classid,
                        String data,
                        String type,
                        Param[] params)
      throws VXMLException;
}
```

```
package com.genmagic.invui.vxml.tags;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.dom.Object;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.dom.Param;
import com.genmagic.invui.vxml.interp.TagHandler;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLException;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.ObjectHandler;
import com.genmagic.invui.vxml.interp.Util;
public class ObjectTag implements TagHandler {
   public void perform(VXMLInterp ip, Element self) throws VXMLException {
      Object obj = (Object)self;
      String id = obj.getClassid();
      String codebase = obj.getCodebase();
      String codetype = obj.getCodetype();
      String data = obj.getData();
      String type = obj.getType();
      String archive = obj.getArchive();
      if (id == null) {
         throw new EventException("error.semantic",
                        "classid is not specified");
      }
      if (codetype == null) {
         codetype = type;
      }
      Param[] params = obj.getParams();
      ip.setVariable(obj.getName(), Boolean.TRUE);
      ObjectHandler handler = Util.getObjectHandler(id);
/*
      ObjectHandler handler = ip.getObjectHandler(id,
                                 codebase,
                                 codetype,
                                 archive);
*/
      if (handler == null) {
         throw new EventException("error.semantic",
                        "could not found object handler for :" +
                        id);
      }
      java.lang.Object result = handler.handle(ip,
                                 id,
                                 data,
                                 type,
                                 params);
      ip.setVariable(obj.getName(), result);
   }
}
```

```
package com.genmagic.invui.vxml.dialogengine;
class OutgoingCallRequest {
   boolean waiting = true;
   boolean connected = false;
   boolean cancelled = false;
   synchronized boolean callConnected() {
      if (waiting == false) {
         return false;
      }
      connected = true;
      notify();
      return true;
   }
   synchronized boolean callCancelled() {
      if (waiting == false) {
         return false;
      }
      cancelled = true;
      notify();
      return true;
   }
   synchronized boolean waitForConnection(long timeout) {
      long until = System.currentTimeMillis();
      long timeLeft = timeout;
      while(cancelled==false &&
            connected==false &&
            timeout > 0) {
         try {
            wait(timeout);
         } catch (InterruptedException ex) {
         }
         timeout = until - System.currentTimeMillis();
      }
      waiting = false;
      return connected;
   }
};
```

```
package com.genmagic.invui.vxml.dom;
import com.genmagic.invui.vxml.interp.EventException;
/**
 * @since Vxml1.0
 */
abstract public class Param extends VXMLElement {
   public Param() {
      super("param");
   }
   abstract public String getName();
   abstract public Expr getExpr() throws EventException;
   abstract public String getValue();
   abstract public String getValueType();
   abstract public String getType();
}
```

```
package com.genmagic.invui.vxml.tags;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.dom.*;
import com.genmagic.invui.vxml.interp.*;
public class PrintTag implements TagHandler {
   public void perform(VXMLInterp ip, Element self) throws VXMLException {
      String msg = self.getAttribute("msg");
      if (msg.length()>0) {
         System.out.println(msg);
      } else {
         String expr = self.getAttribute("expr");
         System.out.println(ip.evalExpression(new ExprImpl(expr)));
      }
   }
}
```

Document ID: 5

```
package com.genmagic.invui.vxml.dom;
import com.genmagic.invui.vxml.interp.EventException;
abstract public class Prompt extends VXMLElement {
   public Prompt() {
      super("prompt");
   }
   /**
    * Get timeout
    */
   abstract public int getTimeout();
   /**
    * Get bargein
    */
   abstract public int getBargein();
   /**
    * Get count. Default is 1
    */
   abstract public int getCount();
   /**
    * @since Vxml1.0
    */
   abstract public Expr getCond() throws EventException;
}
```

```
package com.genmagic.invui.vxml.interp;
import com.genmagic.invui.vxml.prompt.CompositePrompt;
public interface Promptable {
   public void appendPrompt(String klass, CompositePrompt set) throws EventException;
}
```

Document Page

```
package com.genmagic.invui.vxml.prompt;
/**
 * PromptGenerator is the parent interface for all prompt generator clasess.
 * Specific implementation of PromptGenerator will interpret
 * the given content and append the converted prompts to the given
 * composite prompt.
 */
public interface PromptGenerator {
   /**
    * Interpret the content and append the converted prompts to
    * the given composite prompt.
    * @param composite the compostie prompt to which the generated
    *       prompts will be appended.
    * @param content the content to be converted to prompts.
    * @exception InvalidFormatException if the format of the content
    *           is invalid.
    */
   public void appendPrompt(CompositePrompt composite, String content) throws InvalidFormatException;
}
```

Document ID: 7/9/2002 - 10:26:15 PM

```
package com.genmagic.invui.vxml.prompt;
import java.util.Hashtable;
import com.genmagic.invui.util.Log;
import com.genmagic.invui.util.LogFactory;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.Util;
/**
 * PromptGeneratorFactory is the object which is responsible for
 * finding appropriate PromptGenerator object for given type.
 * The default behavior is that if it cannot find a generator object
 * in the cache, it tries to create a PromptGenerator object whose
 * class name is
 * <code>
 * com.genmagic.invui.vxml.prompt.generator.<type_name>.Handler
 * </code>
 * where <code>type_name</code> is the name of type.
 * It is also possible to customize the way it creates the generator
 * by replacing the default factory with your PromptGeneratorFactory
 * object. (setPromptGeneratorFactory)
 */
abstract public class PromptGeneratorFactory {
  static private Log log = LogFactory.getLogFor("ppt");
  static private Hashtable generators = new Hashtable();
  static private PromptGeneratorFactory _factory = new DefaultFactory();
  static {
    generators.put("string", new PromptGenerator() {
      public void appendPrompt(CompositePrompt set, String content) {
        set.appendTTS(content);
      }
    });
    generators.put("number_mid",
            new com.genmagic.invui.vxml.prompt.generator.number.Handler("mid"));
    generators.put("phone_mid",
            new com.genmagic.invui.vxml.prompt.generator.phone.Handler("mid"));
    generators.put("digits_mid",
            new com.genmagic.invui.vxml.prompt.generator.digits.Handler("mid"));
    generators.put("alphanum_mid",
            new com.genmagic.invui.vxml.prompt.generator.alphanum.Handler("mid"));
    generators.put("date_mid",
            new com.genmagic.invui.vxml.prompt.generator.date.Handler("mid"));
    generators.put("dollar_mid",
            new com.genmagic.invui.vxml.prompt.generator.dollar.Handler("mid"));
    generators.put("time_mid",
            new com.genmagic.invui.vxml.prompt.generator.time.Handler("mid"));
    generators.put("duration_mid",
            new com.genmagic.invui.vxml.prompt.generator.duration.Handler("mid"));
    String custom_types = System.getProperty("jde.prompt.types");
    if (custom_types != null) {
      String[] types = Util.toArray(custom_types);
      for (int i=0; i<types.length; i++) {
        String t = types[i];
        log.info("loading custom prompt generator :" + t );
        String file =
          System.getProperty("jde.prompt." + t);
        try {
          PromptGenerator generator =
```

```
                new CustomPromptGenerator(new java.io.FileInputStream(file));
              generators.put(t, generator);
            } catch (Exception ex) {
              log.info("fail to load custom prompt generator :" + ex.getMessage());
            }
          }
        }
      }
      /**
       * Set the customized factory
       * @param factory the user defined prompt generator factory object.
       */
      static public void setPromptGeneratorFactory(PromptGeneratorFactory factory) {
        _factory = factory;
      }
      /**
       * Register the prompt generator with given name. It will replace
       * if there already exists one.
       * @param klass the type of generator. (date/time/phone etc.)
       * @param gen the prompt generator used for given type.
       */
      static public void register(String klass, PromptGenerator gen) {
        generators.put(klass, gen);
      }
      /**
       * return the prompt generator for given klass(type).
       * It automatically create one if no generator exists.
       * @param klass the type of generator. (date/time/phone etc.)
       * @exception EventException(error.semantic) if it could not find
       *          the appropriate one.
       */
      static public PromptGenerator getPromptGenerator(String klass) throws EventException {
        PromptGenerator gen = (PromptGenerator)generators.get(klass);
        if (gen == null) {
          synchronized(PromptGeneratorFactory.class) {
            gen = (PromptGenerator)generators.get(klass);
            if (gen == null) {
              gen = _factory.createPromptGenerator(klass);
              register(klass, gen);
            }
          }
        }
        return gen;
      }
      /**
       * Abstract method to create a prompt generator object for the given type.
       * This is supposed to be implemented by subclass.
       * @param klass the type of generator to create
       */
      abstract protected PromptGenerator createPromptGenerator(String klass) throws EventException;
      /**
       * Default prompt generator factory. It tires to create object of the
       * class
       * <code>
       *  com.genmagic.invui.vxml.prompt.generator.<klass>.Handler
       * </code>
```

```
 */
static private class DefaultFactory extends PromptGeneratorFactory {
  protected PromptGenerator createPromptGenerator(String klass) throws EventException {
    String klassname =
      "com.genmagic.invui.vxml.prompt.generator." + klass + ".Handler";
    try {
      Class cls = Class.forName(klassname);
      Object obj = cls.newInstance();
      if (obj instanceof PromptGenerator == false) {
        throw new EventException("error.semantic",
                    "not converter class :" +
                    klassname);
      } else {
        return (PromptGenerator)obj;
      }
    } catch (ClassNotFoundException ex) {
      throw new EventException("error.semantic",
                  "converter not found :" +
                  klassname);
    } catch (InstantiationException ex) {
      throw new EventException("error.semantic",
                  "cannot instantiate converter :"+
                  klassname);
    } catch (IllegalAccessException ex) {
      throw new EventException("error.semantic",
                  "cannot instantiate converter :"+
                  klassname);
    }
  }
 }
}
```

```
package com.genmagic.invui.vxml.prompt;
import java.util.Vector;
import java.util.Enumeration;
abstract public class PromptSet implements java.io.Serializable {
   protected Vector _prompts = new Vector();
   transient private Vector _cache = null;
   synchronized public void setAddPrompt(String promptname) {
      _cache = null;
      _prompts.addElement(promptname);
   }
   synchronized public void addPrompts(String []prompts) {
      _cache = null;
      for (int i=0; i<prompts.length; i++) {
         _prompts.addElement(prompts[i]);
      }
   }
   public String getPrompts() {
      return toVXML();
   }
   abstract protected Vector construct();
   protected static java.util.Random rand = new java.util.Random();
   public String toVXML() {
      if (_cache == null) {
         synchronized (this) {
            if (_cache == null) {
               _cache = construct();
            }
         }
      }
      return (String)_cache.elementAt( (int)(rand.nextFloat() * _cache.size()));
   }
   public static void main(String a[]) throws Exception {
      Random r = new Random();
      r.setAddPrompt("a.wav");
      r.setAddPrompt("b.wav");
      r.setAddPrompt("c.wav");
      r.setAddPrompt("d.wav");
      r.setAddPrompt("e.wav");
      r.setAddPrompt("f.wav");
      r.setAddPrompt("g.wav");
      r.toVXML();
      System.out.println(r.toVXML());
   }
}
```

```
package com.genmagic.invui.vxml.tags;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.dom.Prompt;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.interp.Properties;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLException;
import com.genmagic.invui.vxml.interp.VXMLConstants;
import com.genmagic.invui.vxml.interp.TagHandler;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.SpeechControl;
public class PromptTag implements TagHandler, VXMLConstants {
   public void perform(VXMLInterp ip, Element self) throws VXMLException {
      Prompt p = (Prompt)self;
      Expr cond = p.getCond();
      int count = p.getCount();
      SpeechControl speech = ip.getSpeechControl();
      if ( (cond == null ||
          ip.evalBooleanExpression(cond)) &&
         speech.checkCount(count)) {
         int b = p.getBargein();
         Properties props = ip.getCurrentProperties();
         boolean bargein = (b == BARGEIN_NONE) ?
            props.getBooleanProperty("bargein", true) :
            (b == BARGEIN_ON);
         boolean backup = speech.getBargein();
         speech.setBargein(bargein);
         int timeout = p.getTimeout();
         if (timeout >= 0) {
            speech.setTimeout(timeout);
         }
         ip.evalChildren(self);
         speech.setBargein(backup);
      }
   }
}
```

```
package com.genmagic.invui.vxml.interp;
public interface Properties {
   public int getIntProperty(String name, int defValue);
   public long getTimeProperty(String name, long defValue);
   public String getProperty(String name, String defValue);
   public boolean getBooleanProperty(String name, boolean defValue);
   public void setProperty(String name, java.lang.Object obj) throws VXMLException;
}
```

Document Evaluation Copy

```
package com.genmagic.invui.vxml.property;
import java.util.Hashtable;
import java.util.Vector;
import java.util.Enumeration;
import com.genmagic.invui.vxml.interp.AssertException;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.RuntimeEventException;
import com.genmagic.invui.vxml.interp.Properties;
import com.genmagic.invui.vxml.interp.VXMLException;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.Util;
public class PropertiesImpl extends Hashtable implements Properties {
  static Hashtable _listenerTable = new Hashtable();
  static {
    addPropertyChangeListener("bargein", new BargeinPropertyListener());
    addPropertyChangeListener("searchpath", new SearchPathPropertyListener());
    addPropertyChangeListener("nuance.toplevelgrammar", new RootGrammarPropertyListener());
    addPropertyChangeListener("grammarlist", new GrammarListPropertyListener());
  }

  static void addPropertyChangeListener(String name, PropertyChangeListener l) {
    _listenerTable.put(name, l);
  }
  static PropertyChangeListener getPropertyChangeListener(String name) {
    return (PropertyChangeListener)_listenerTable.get(name);
  }
  private PropertiesImpl _parent;
  private VXMLInterp _interp;
  Vector listeners = null;
  public static final Properties emptyProperties = new Properties() {
    public void setProperty(String s, Object obj) {
      throw new AssertException("property cannot be set to empty impl :"+
                s + "," + obj);
    }
    public String getProperty(String n, String defValue) {
      return defValue;
    }
    public int getIntProperty(String n, int defValue) {
      return defValue;
    }
    public long getTimeProperty(String n, long defValue) {
      return defValue;
    }
    public boolean getBooleanProperty(String n, boolean defValue) {
      return defValue;
    }
  };
  public PropertiesImpl(VXMLInterp ip) {
    _interp = ip;
  }
  public PropertiesImpl(PropertiesImpl parent) {
    _parent = parent;
    _interp = _parent._interp;
  }
  public void setProperty(String s, Object obj) throws VXMLException {
    // System.out.println("CREATING property " + s + " value = " + obj);
```

```
        super.put(s, obj);
        PropertyChangeListener l = getPropertyChangeListener(s);
        if (l != null) {
            if (listeners == null) {
                listeners = new Vector();
            }
            listeners.addElement(l);
            // System.out.println("NOTIFYING property " + s + " value=" + obj);
            l.propertyAdded(_interp, obj);
        }
    }
    public void dispose() throws VXMLException {
        if (listeners != null) {
            Enumeration e = listeners.elements();
            while(e.hasMoreElements()) {
                // System.out.println("REMOVING property ");
                ((PropertyChangeListener)e.nextElement()).propertyRemoved(_interp);
            }
        }
    }
    public String getProperty(String n, String defValue) {
        Object value = getProperty(n);
        return value == null ? defValue : value.toString();
    }
    public int getIntProperty(String n, int defValue) {
        Object value = getProperty(n);
        if (value == null) {
            return defValue;
        } else if (value instanceof Number) {
            return ((Number)value).intValue();
        } else {
            try {
                int ivalue = Integer.parseInt(value.toString());
                super.put(n, new Integer(ivalue));
                return ivalue;
            } catch (NumberFormatException ex) {
                throw new RuntimeEventException("error.semantic",
                                    "cannot convert to int value :" + value.toString());
            }
        }
    }
    public long getTimeProperty(String n, long defValue) {
        Object value = getProperty(n);
        if (value == null) {
            return defValue;
        } else if (value instanceof Number) {
            return ((Number)value).longValue();
        } else {
            try {
                long lvalue = Util.parseTime(value.toString());
                super.put(n, new Long(lvalue));
                return lvalue;
            } catch (NumberFormatException ex) {
                throw new RuntimeEventException("error.semantic",
                                    "cannot convert to time value :" + value.toString());
            }
```

```
    }
  }
  public boolean getBooleanProperty(String n, boolean defValue) {
    Object value = getProperty(n);
    if (value == null) {
      return defValue;
    } else if (value instanceof Boolean) {
      return ((Boolean)value).booleanValue();
    } else {
      Boolean b = Boolean.valueOf(value.toString());
      super.put(n, b);
      return b.booleanValue();
    }
  }
  private Object getProperty(String n) {
    Object obj = super.get(n);
    return (obj == null && _parent != null) ?
      _parent.getProperty(n) : obj;
  }
  public PropertiesImpl getParent() {
    return _parent;
  }
  private int depth = 0;
  public void up() {
    depth ++;
  }
  public void down() {
    depth --;
  }
  public int depth() {
    return depth;
  }
  public String toString() {
    return "Property: depth = " + depth + "\n" + super.toString();
  }
}
```

```
package com.genmagic.invui.vxml.dom;
import com.genmagic.invui.vxml.interp.EventException;
/**
 * @since Vxml1.0
 */
abstract public class Property extends VXMLElement {
   protected Property() {
      super("property");
   }
   abstract public String getName();
   abstract public String getValue();
   /**
    * GMI Extension
    */
   abstract public Expr getExpr() throws EventException;
}
```

```
package com.genmagic.invui.vxml.property;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLException;
public interface PropertyChangeListener {
   public void propertyAdded(VXMLInterp ip, Object value) throws VXMLException;
   public void propertyRemoved(VXMLInterp ip) throws VXMLException;
};
```

Document Page - Unpublished

```
package com.genmagic.invui.vxml.tags;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.dom.Property;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLException;
import com.genmagic.invui.vxml.interp.VXMLConstants;
import com.genmagic.invui.vxml.interp.TagHandler;
import com.genmagic.invui.vxml.interp.EventException;
public class PropertyTag implements TagHandler, VXMLConstants {
   public void perform(VXMLInterp ip, Element self) throws VXMLException {
      Property p = (Property)self;
      String name = p.getName();
      if (name == null) {
         throw new EventException("error.semantic",
                        "no name specified in property");
      }
      Object value = p.getValue();
      if (value == null) {
         Expr expr = p.getExpr();
         if (expr != null) {
            value = ip.evalExpression(expr);
         }
         if (value == null) {
            throw new EventException("error.semantic",
                           "no value specified in property");
         }
      }
      ip.setProperty(name, value);
   }
}
```

```
package com.genmagic.invui.vxml.prompt;
import java.util.Vector;
import java.util.Enumeration;
public class Random extends RandomPrompts {
   public static void main(String a[]) throws Exception {
      Random r = new Random();
      r.setAddPrompt("a.wav");
      r.setAddPrompt("b.wav");
      r.setAddPrompt("c.wav");
      r.setAddPrompt("d.wav");
      r.setAddPrompt("e.wav");
      r.setAddPrompt("f.wav");
      r.setAddPrompt("g.wav");
      r.toVXML();
      System.out.println(r.toVXML());
   }
}
```

Document1 - Microsoft Word

```
package com.genmagic.invui.vxml.prompt;
import java.util.Vector;
import java.util.Enumeration;
public class RandomPrompts extends PromptSet {
  synchronized protected Vector construct() {
    int size = _prompts.size();
    Vector cache = new Vector();
    int base = 0;
    int six  = size % 6;
    int five = size % 5;
    int four = size % 4;
    int min = Math.min(six, five);
    int max = Math.max(six, five);
    if (size <= 8) {
      min = Math.min(min, four);
      max = Math.max(max, four);
    }
    if (min == 0) {
      if (min == six) {
        base = 6;
      } else if (min == five) {
        base = 5;
      } else if (min == four) {
        base = 4;
      }
    } else {
      if (max == six) {
        base = 5;
      } else if (max == five) {
        base = 4;
      } else if (max == four) {
        base = 3;
      }
    }
    int rem = size % base;
    int num = (size - rem) / base;

    for (int i=0; i<size; ) {
      int n = base;
      if (rem-->0) n++;
      StringBuffer buf = new StringBuffer();
      buf.append("<var name='tmp' expr='Math.random()*" +
            n + "'/>\n");
      buf.append("<if cond='1.0>=tmp'>\n");
      buf.append("  <audio src='builtin:" + _prompts.elementAt(i) + "'/>\n");
      i++;
      for (int j=1; j < n && i< size; j++, i++) {
        buf.append("<elseif cond='" + (j+1) +".0>=tmp'/>\n");
        buf.append("  <audio src='builtin:" + _prompts.elementAt(i) + "'/>\n");
      }
      buf.append("</if>");
      cache.addElement(buf.toString());
    }
    if (cache.size()==0) {
      cache.addElement("");
    }
```

```
        return cache;
    }
    public static void main(String a[]) throws Exception {
        RandomPrompts r = new RandomPrompts();
        r.setAddPrompt("a.wav");
        r.setAddPrompt("b.wav");
        r.setAddPrompt("c.wav");
        r.setAddPrompt("d.wav");
        r.setAddPrompt("e.wav");
        r.setAddPrompt("f.wav");
        r.setAddPrompt("g.wav");
        r.toVXML();
        System.out.println(r.toVXML());
    }
}
```

```
package com.genmagic.invui.vxml.dom;
import org.w3c.dom.Node;
import com.genmagic.invui.vxml.interp.EventException;
abstract public class Record extends FormItem {
   protected Record() {
      super("record");
   }
   /**
    * Type of mime type of record
    */
   abstract public String getType();
   /**
    *
    */
   abstract public int getFinalSilence();
   /**
    * @deprecated as of Vxml1.0
    */
   abstract public int getMaxLength();
   /**
    * @since Vxml1.0
    */
   abstract public int getMaxTime();
   /**
    *
    */
   abstract public String getDTMFTerm();
   /**
    *
    */
   abstract public boolean getBeep();
   /**
    * @since Vxml1.0
    */
   abstract public boolean getModal();
   /**
    * Returns prompt related children only.
    * prompt, audio, PCDATA, value, enumerate
    */
   abstract public Node[] getPromptElements();
}
```

```
package com.genmagic.invui.vxml.tags;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.prompt.CompositePrompt;
import com.genmagic.invui.vxml.dom.Record;
import com.genmagic.invui.vxml.dom.Catch;
import com.genmagic.invui.vxml.dom.Filled;
import com.genmagic.invui.vxml.interp.*;
public class RecordTag implements TagHandler, VXMLConstants {
  public void perform(VXMLInterp ip, Element self) throws VXMLException {
    Record record = (Record)self;
    SpeechControl speech = ip.getSpeechControl();
    Properties props = ip.getCurrentProperties();
    int maxtime = record.getMaxTime();
    if (maxtime <0) {
      maxtime = (int)props.getTimeProperty("maxtime",
                                DEFAULT_RECORD_MAX_LENGTH);
    }
    int finalsilence = record.getFinalSilence();
    if (finalsilence <0) {
      finalsilence = (int)props.getTimeProperty("finalsilence",
                                  200);
    }
    int timeout = speech.getTimeout();
    if (timeout <0) {
      timeout = (int)props.getTimeProperty("timeout",
                                DEFAULT_RECORD_TIMEOUT);
    }
    speech.clearTimeout();
    XMLLoader loader = ip.getAppContext().getLoader();
    String stateName =
      loader.getDocumentBase() +
      loader.getFile() + "#" +
      record.getStateName();
    // speech.setReprompt(true);
    // speech.backupPrompts();
    String varName = record.getName();
    // EventCounter evc = new EventCounter();
    try {
      // REMIND: This may be needed if we enables recognition.
      // speech.enterState(stateName);
      if (speech.getReprompt()) {
        // Note: This may have to be incremented at "reprompt"
        ip.eval(record.getPromptElements());
        speech.setReprompt(false);
      }
      if (record.getBeep()) {
        // speech.appendSilence(300); // for the time being.
        speech.appendDTMF("#"); // for the time being.
      }
      speech.playPrompt();
      RecordVar recvar = speech.record(timeout,
                          maxtime,
                          finalsilence);
      ip.debug("set var name=" + varName + " val=" +
            recvar);
```

```
            ip.setVariable(varName, recvar);
            ip.createVariable(varName + "$.duration",
                        new Integer(recvar.getDuration()));
/*
            ip.setVariable(varName + "$.size",
                        new Integer(recvar.getSize()));
            ip.setVariable(varName + "$.termchar", );
*/
            Filled filled[] = record.getParentForm().getFilled(new String[]{varName}, ip);
            if (filled != null) {
               ip.eval(filled);
            }
         } finally {
            // speech.leaveState();
         }
      }
   }
```

Document ID: 1

```
package com.genmagic.invui.vxml.interp;
import java.io.*;
import com.genmagic.invui.vxml.prompt.CompositePrompt;
import com.genmagic.invui.util.http.MimePart;
public class RecordVar implements Promptable, MimePart {
   private String filename;
   private String local;
   private int duration;
   public RecordVar(String ext, String l, int d) {
      filename = ext;
      local = l;
      duration = d;
   }
   public String getContentDispositionParameters() {
      return "; filename=\"" + filename + "\"";
   }
   public String getContentType() {
      return "audio/x-wav";
   }
   public void writeTo(java.io.OutputStream out) throws IOException {
      if (local != null) {
         InputStream in = new FileInputStream(local);
         try {
            byte[] buf = new byte[1024];
            int len;
            while ( (len=in.read(buf)) >=0) {
               out.write(buf, 0, len);
            }
         } finally {
            in.close();
         }
      }
   }
   public int getDuration() {
      return duration;
   }
   public void appendPrompt(String klass, CompositePrompt set) throws EventException {
      if ("string".equals(klass)) {
         set.appendTTS(filename);
      } else if (klass == null ||
            "audio".equals(klass)) {
         set.appendAudio(filename, null);
      } else {
         Util.appendPrompt(toString(), klass, set);
      }
   }
   public String toString() {
      return filename;
   }
}
```

```
package com.genmagic.invui.vxml.interp;
import java.util.Enumeration;
abstract public class RecResult {
   protected Grammar activated = null;
   public Grammar getActivatedGrammar() {
      return activated;
   }
   public void performAction(Object cxt) throws Exception {
      activated.performAction(this, cxt);
   }
   abstract public SlotResult getResult(String name) throws EventException;
   abstract public SlotResult getRawResult(String name) throws EventException;
   abstract public String getActiveGrammarName() throws EventException;
   abstract public int getNumSlots();
   abstract public Enumeration getSlotNames();
}
```

```
package com.genmagic.invui.vxml.dialogengine;
import java.util.Enumeration;
import java.util.Vector;
import com.genmagic.invui.vxml.idl.invui.*;
import com.genmagic.invui.vxml.idl.invui.SlotPackage.*;
import com.genmagic.invui.vxml.interp.Grammar;
import com.genmagic.invui.vxml.interp.RecResult;
import com.genmagic.invui.vxml.interp.SlotResult;
import com.genmagic.invui.vxml.interp.EventException;
class RecResultImpl extends RecResult {
   private RecognitionResults _results;
   public RecResultImpl(RecognitionResults r) {
      _results = r;
   }
   void setActiveGrammar(Grammar g) {
      activated = g;
   }
   public int getNumSlots() {
      if (_results.results.length == 0) {
         return 0;
      }
      return _results.results[0].nl_interpretations.length;
   }
   public String getActiveGrammarName() throws EventException {
      return _results.results[0].nl_interpretations[0].grammar;
   }

   public SlotResult getResult(String name) throws EventException {
      return activated.getSlotResult(this, name);
   }
   public SlotResult getRawResult(String name) throws EventException {
      Slot[] s = _results.results[0].nl_interpretations[0].slots;
      String utterance = _results.results[0].result;
          if (utterance == null) { // should not happen
         utterance= "";
      }
      for (int i =0; i<s.length; i++) {
         switch(s[i]._discriminator().value()) {
         case SlotType._T_String:
            sslot_s ss = s[i].sslot();
            if (name.equals(ss.name)) {
               return new SlotResult(ss.value,
                              ss.confidence,
                              utterance);
            }
            break;
         case SlotType._T_Int:
            islot_s is = s[i].islot();
            if (name.equals(is.name)) {
               return new SlotResult(new Integer(is.value),
                              is.confidence,
                              utterance);
            }
            break;
         case SlotType._T_Structure:
            rslot_s rs = s[i].rslot();
```

```
            // copy(slots, rs.child_slot);
            break;
          }
        }
        return null;
      }
      public Enumeration getSlotNames() {
        Vector names = new Vector();
        Slot[] s = _results.results[0].nl_interpretations[0].slots;
        for (int i =0; i<s.length; i++) {
          switch(s[i]._discriminator().value()) {
          case SlotType._T_String:
            names.addElement(s[i].sslot().name);
            break;
          case SlotType._T_Int:
            names.addElement(s[i].islot().name);
            break;
          case SlotType._T_Structure:
            break;
          }
        }
        return names.elements();
      }
      public String toString() {
        StringBuffer b = new StringBuffer();

        Slot[] s = _results.results[0].nl_interpretations[0].slots;
        for (int i =0; i<s.length; i++) {
          switch(s[i]._discriminator().value()) {
          case SlotType._T_String:
            sslot_s ss = s[i].sslot();
            b.append('<');
            b.append(ss.name);
            b.append(',');
            b.append(ss.value.toString());
            b.append("> ");
            break;
          case SlotType._T_Int:
            islot_s is = s[i].islot();
            b.append('<');
            b.append(is.name);
            b.append(',');
            b.append(String.valueOf(is.value));
            b.append("> ");
            break;
          case SlotType._T_Structure:
            rslot_s rs = s[i].rslot();
            // copy(slots, rs.child_slot);
            break;
          }
        }
        return b.toString();
      }
    }
```

```
package com.genmagic.invui.vxml.dom;
abstract public class Reprompt extends VXMLElement {
   public Reprompt() {
      super("reprompt");
   }
}
```

Document 7 of 13

```
package com.genmagic.invui.vxml.tags;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.dom.Clear;
import com.genmagic.invui.vxml.interp.TagHandler;
import com.genmagic.invui.vxml.interp.VXMLInterp;
public class RepromptTag implements TagHandler {
   public void perform(VXMLInterp ip, Element self) {
      // increment prompt count
      ip.getSpeechControl().setReprompt(true);
   }
}
```

Oracle Exhibit 2015, Page 5

```
package com.genmagic.invui.vxml.interp;
import java.io.*;
public interface ResourceManager {
   public class TmpFile {
      public String internal;
      public String external;
   }
   public String get(String path);
   public String save(String path, long expire, InputStream in) throws IOException;
   public TmpFile createTmpFile(String name, String suf);
}
```

Document Page

```
package com.genmagic.invui.vxml.dialogengine;
import java.io.*;
import java.util.Hashtable;
import com.genmagic.invui.util.Cache;
import com.genmagic.invui.vxml.interp.ResourceManager;
public class ResourceManagerFileImpl implements ResourceManager {
   private static String remote_base = System.getProperty("jde.cache.basepath.remote", "\\\\moshimapc\\temp\\");
   private static String local_base = System.getProperty("jde.cache.basepath.local", "c:\\temp\\");
   static {
      File f = new File(local_base);
      if (f.isDirectory()) {
         File[] list = f.listFiles(new FileFilter() {
            public boolean accept(File f) {
               return f.isFile();
            }
         });
         for (int i=0; i<list.length; i++) {
            list[i].delete();
         }
      }
   }
   //
   // temporaly solution
   //
   static class CacheItem implements Cache.Item {
      String name;
      String fname;
      String cache;
      int status = 0;
      long expire;
      CacheItem(String n, String f, String c, long e) {
         name = n;
         fname = f;
         cache = c;
         expire = e;
      }
      public Object getName() {
         return name;
      }
      synchronized void completed() {
         status = 1;
         notify();
      }
      synchronized void failed() {
         status = -1;
         notify();
      }
      synchronized String get() {
         while (status == 0) {
            try {
               wait();
            } catch (InterruptedException ex) {}
         }
         if (status == -1) {
            return null;
         }
```

```
        return fname;
    }
    public int hashCode() {
        return name.hashCode();
    }
    public boolean equals(Object o) {
        if (o instanceof CacheItem) {
            return name.equals(((CacheItem)o).name);
        }
        return false;
    }
    public boolean isExpired() {
        return expire > 0 ?
            (expire < System.currentTimeMillis()) : false;
    }
    public boolean dispose() {
        return new File(cache).delete();
    }
}
private int index = 0;
private Cache cache = new Cache();
public String get(String path) {
    CacheItem c = (CacheItem)cache.getItem(path);
    if (c != null) {
        return c.get();
    } else {
        return null;
    }
}
public ResourceManager.TmpFile createTmpFile(String name, String suf) {
    ResourceManager.TmpFile tmp = new ResourceManager.TmpFile();
    tmp.external = remote_base;
    tmp.internal = local_base;
    synchronized(this) {
        String t = name + (index++) + suf;
        tmp.external = remote_base + t;
        tmp.internal = local_base + t;
    }
    return tmp;
}
public String save(String path, long expire, InputStream in) throws IOException {
    ResourceManager.TmpFile tmp = null;
    CacheItem i;
    synchronized (this) {
        tmp = createTmpFile("cache", ".wav");
        i = new CacheItem(path, tmp.external, tmp.internal, expire);
        cache.add(i);
    }
    boolean success = false;
    try {
        OutputStream out = new FileOutputStream(tmp.internal);
        byte[] b = new byte[512];
        int len;

        while ( (len = in.read(b)) >=0) {
            out.write(b, 0, len);
```

```
            }
            try {
               in.close();
            } finally {
               out.close();
            }
            success = true;
         } finally {
            if (success) {
               i.completed();
            } else {
               cache.removeItem(path);
               i.failed();
            }
         }
         return tmp.external;
      }
   }
```

Document #1 (2002)

```
package com.genmagic.invui.vxml.dom;
import com.genmagic.invui.vxml.interp.EventException;
/**
 * @since Vxml1.0
 */
abstract public class Return extends VXMLElement {
   public Return() {
      super("return");
   }
   /**
    */
   abstract public String getEvent();
   /**
    */
   abstract public String[] getNameList();
}
```

```
package com.genmagic.invui.vxml.property;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.VXMLException;
public class RootGrammarPropertyListener implements PropertyChangeListener {
   public void propertyAdded(VXMLInterp ip, Object value) throws VXMLException {
      ip.getSpeechControl().setParameter("Nuance.TopLevelGrammar", value.toString());
   }
   public void propertyRemoved(VXMLInterp ip) throws VXMLException {
      String previous =
         ip.getCurrentProperties().getProperty("nuance.toplevelgrammar", ".DynamicTop");
      ip.getSpeechControl().setParameter("Nuance.TopLevelGrammar", previous.toString());
   }
};
```

Oracle Ex. 2010, p. 2 of 5

```
package com.genmagic.invui.vxml.interp;
import org.w3c.dom.Node;
public class RuntimeEventException extends RuntimeException {
   String eventName;
   //Node source;
   public RuntimeEventException(String n) {
      eventName = n;
   }
   public RuntimeEventException(String n, String msg) {
      super(msg);
      eventName = n;
   }
/*
   public RuntimeEventException(String n, String msg, Node s) {
      super(msg);
      eventName = n;
      source = s;
   }
   public Node getSource() {
      return source;
   }
*/
   public String getMessage() {
      return eventName + " : " + super.getMessage();
   }
   public String getEventName() {
      return eventName;
   }
}
```

```
package com.genmagic.invui.vxml.dom;
abstract public class Sayas extends VXMLElement {
   public Sayas() {
      super("sayas");
   }
   /**
    * Get type to which this variable will be converted
    * default is "string"?
    */
   abstract public String get_Class();
}
```

Document ID: 20030205

```
package com.genmagic.invui.vxml.tags;
import java.lang.Object;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.prompt.CompositePrompt;
import com.genmagic.invui.vxml.dom.Sayas;
import com.genmagic.invui.vxml.interp.TagHandler;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.VXMLException;
import com.genmagic.invui.vxml.interp.Util;
public class SayasTag implements TagHandler {
   public void perform(VXMLInterp ip, Element self) throws VXMLException {
      Sayas sayas = (Sayas)self;
      String klass = sayas.get_Class();
      String content = "";
      // may needs to be optimized
      if (sayas.hasChildNodes()) {
         CompositePrompt backup = ip.getCompositePrompt();
         //Prompt[] backup = ip.getPrompts();
         ip.clearPrompts();
         ip.evalChildren(self);
         CompositePrompt p = ip.getCompositePrompt();
         //Prompt[] p = ip.getPrompts();
         ip.clearPrompts();
         backup.appendTo(ip);
         //ip.appendPrompts(backup);
         content = p.getAllTTS();
         //content = Util.toString(p);
      }
      Util.appendPrompt(content, klass, ip);
   }
}
```

```
package com.genmagic.invui.vxml.dom;
/**
 * @since Vxml1.0
 */
abstract public class Script extends VXMLElement implements Fetchable {
   public Script() {
      super("script");
   }
   abstract public String getSrc();
   abstract public String getCharset();
}
```

```
package com.genmagic.invui.vxml.script;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.dom.Expr;
public interface ScriptRuntime {
   public void init(VXMLInterp ip);
   public void destroy();
   public Object getUndefinedValue();
   public boolean isNotFound(Object obj);
   /*=======================================
    * Variable
    *=======================================*/
   /**
    * Checks if the variable of given name exists in given scope.
    * @param scope the scope of variable to check
    * @param name the name of variable to check
    * @return true if the variable exists in the scope.
    */
   abstract public boolean hasDefinedVariable(int scope, String name);
   /**
    * Gets the value of the variable.
    * @param name the name of variable
    * @return a value of the variable
    * @exception EventException of error.semantic if no variable exists,
    *          or undefined.
    */
   abstract public Object getVariable(String name) throws EventException;
   /**
    * Assigns a value to a variable.
    * @param name the name of variable to assign to.
    * @param value the value to be assigned to the variable
    * @exception EventException with "error.semantic" if variable does not
    *          exist.
    */
   abstract public void setVariable(String name, Object value) throws EventException;
   /**
    * Creates a new variable with the given value in the current scope.
    * @param name the name of variable to create.
    * @param value the value to be assigned to the new variable
    */
   abstract public void createVariable(String name, Object value) throws EventException ;
   /**
    * Creates a new undefined variable in the current scope.
    * @param name the name of variable to create.
    */
   abstract public void createVariable(String name) throws EventException ;
   /**
    * Creates new variables with the same given value in the current scope.
    * @param names the array of variable names to create.
    * @param value the value to be assigned to the new variables
    */
   abstract public void createVariables(String names[], Object value) throws EventException;
   /**
    * Clear the content of variable. After this, the variable is marked
    * as undefined, and all subsequence access to this variable causes
    * error.semantic event to be thrown.
```

```
 * @param name the variable name to clear
 * @exception name EventException with "error.semantic" if the variable
 *          does not exist.
 */
abstract public void clearVariable(int scope, String name) throws EventException;
/**
 * Clear the content of variables in specific scope
 * After this, variables are marked as undefined, and all subsequence
 * access to those variables causes error.semantic event to be thrown.
 *
 * @param names the variable names to clear
 * @exception name EventException with "error.semantic" if one of variables
 *          does not exist.
 */
abstract public void clearVariables(int scope, String names[]) throws EventException;
/**
 * Remove the variables from the current scope.
 *
 * @param names the variable names to remove
 * @exception name EventException with "error.semantic" if one of variables
 *          does not exist.
 */
abstract public void removeVariable(String name) throws EventException;
/*=========================================
 * Expression
 *=========================================*/
/**
 * Evaluates the expression and returns the result.
 * @param expr the expression object to evaluate
 * @return the result object
 * @return EventException with "error.syntax" if syntax error in
 *                  expression, or "error.semantic" if variable
 *                  used in the expression does not exists, or
 *                  is undefiened.
 */
abstract public Object evalExpression(Expr expr) throws EventException;
/**
 * Evaluates the string and returns the result.
 * @param str the string to evaluate
 * @return the result object
 * @return EventException with "error.syntax" if syntax error in
 *                  expression, or "error.semantic" if variable
 *                  used in the expression does not exists, or
 *                  is undefiened.
 */
abstract public Object evalString(String str) throws EventException;
/**
 * Evaluates the expression and returns the boolean result.
 * @param expr the expression object to evaluate
 * @return the true if the result is boolean and true, false otherwise.
 * @return EventException with "error.syntax" if syntax error in
 *                  expression, or "error.semantic" if variable
 *                  used in the expression does not exists, or
 *                  is undefiened.
abstract public boolean evalBooleanExpression(Expr expr) throws EventException;
 */
```

```
    /*=========================================
     * Scoping
     *=========================================*/
    abstract public Frame getFrame(int scope);
    abstract public void beginApplicationScope();
    abstract public void beginAnonymousScope();
    abstract public void endAnonymousScope();
    abstract public void beginDialogScope();
    abstract public void beginDocumentScope();
    abstract public void enterApplicationRoot();
}
```

Document1 - Microsoft Word

```
package com.genmagic.invui.vxml.script.expr;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Vector;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.interp.ExprImpl;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.AssertException;
import com.genmagic.invui.vxml.interp.VXMLConstants;
import com.genmagic.invui.vxml.script.Frame;
import com.genmagic.invui.vxml.lex.*;
public class ScriptRuntime implements com.genmagic.invui.vxml.script.ScriptRuntime, ExprConstants,
VXMLConstants {
   static private void checkName(String name) {
      if (name.length()==0) {
         throw new AssertException("variable name length should not be 0");
      }
   }
   public void init(VXMLInterp ip) {
      interp = ip;
   }
   public void destroy() {
   }
   public Object getUndefinedValue() {
      return UNDEFINED_OBJ;
   }
   public boolean isNotFound(Object obj) {
      return obj == null;
   }

   public void createVariables(String names[], Object value) {
      for (int i=0; i<names.length; i++) {
         createVariable(names[i], value);
      }
   }
   public void createVariable(String name, Object value) {
      checkName(name);
      Environment env = null;
      char c = name.charAt(0);
      if (c == 's' && name.startsWith("session.")) {
         env = sessionEnv;
         name = name.substring(8);
      } else if (c == 'a' && name.startsWith("application.")) {
         env = appEnv;
         name = name.substring(12);
      } else if (c == 'd' && name.startsWith("document.")) {
         env = docEnv;
         name = name.substring(9);
      } else if (c == 'd' && name.startsWith("dialog.")) {
         env = dialogEnv;
         name = name.substring(7);
      } else if (c == 'l' && name.startsWith("local.")) {
         env = dialogEnv;
         name = name.substring(6);
      } else {
```

```
        env = currentEnv;
      }
      env.createVariable(name, value);
    }
    public void createVariable(String name) {
      createVariable(name, UNDEFINED_OBJ);
    }
    /**
     * Checks if the variable is defined
     */
    public boolean hasDefinedVariable(int scope, String name) {
      checkName(name);
      return getEnv(scope).hasDefinedVariable(name);
    }
    /**
     * Get Variable
     */
    public Object getVariable(String name) throws EventException {
      checkName(name);
      Environment env = null;
      char c = name.charAt(0);
      if (c == 's' && name.startsWith("session.")) {
        env = sessionEnv;
        name = name.substring(8);
      } else if (c == 'a' && name.startsWith("application.")) {
        env = appEnv;
        name = name.substring(12);
      } else if (c == 'd' && name.startsWith("document.")) {
        env = docEnv;
        name = name.substring(9);
      } else if (c == 'd' && name.startsWith("dialog.")) {
        env = dialogEnv;
        name = name.substring(7);
      } else if (c == 'l' && name.startsWith("local.")) {
        env = dialogEnv;
        name = name.substring(6);
      } else {
        /*
         * Implicit
         */
        for (env = currentEnv;
            env != null;
            env = env.getParent()) {
          Object obj = env.getVariable(name);
          if (obj != null) {
            return obj;
          }
        }
        throw new EventException("error.semantic",
                                 "variable not found " + name);
      }
      /*
       * Explicit
       */
      return env.getVariable(name);
    }
```

```
/*
   public Object getVariable(int type, String name) {
      return getEnv(type).getVariable(name);
   }
*/
   public Frame getFrame(int scope) {
      return getEnv(scope);
   }
   protected Environment getEnv(int type) {
      switch (type) {
      case DIALOG:
         return dialogEnv;
      case DOCUMENT:
         return docEnv;
      case APPLICATION:
         return appEnv;
      case SESSION:
         return sessionEnv;
      }
      throw new AssertException("Unknown EnvironmentType " + type);
   }
   public void setVariable(String name, Object value) throws EventException {
      Environment env = null;
      char c = name.charAt(0);
      if (c == 's' && name.startsWith("session.")) {
         env = sessionEnv;
         name = name.substring(8);
      } else if (c == 'a' && name.startsWith("application.")) {
         env = appEnv;
         name = name.substring(12);
      } else if (c == 'd' && name.startsWith("document.")) {
         env = docEnv;
         name = name.substring(9);
      } else if (c == 'd' && name.startsWith("dialog.")) {
         env = dialogEnv;
         name = name.substring(7);
      } else if (c == 'l' && name.startsWith("local.")) {
         env = dialogEnv; // for backward compatibility
         name = name.substring(6);
      } else {
         env = findEnvironment(name);
         if (env == null) {
            throw new EventException("error.semantic",
                                     "variable not found " + name);
         }
      }
      env.setVariable(name, value);
   }
   Environment findEnvironment(String name) {
      for (Environment env = currentEnv;
           env != null;
           env = env.getParent()) {
         if (env.hasVariable(name)) {
            return env;
         }
      }
```

```
    return null;
  }
/*
  public void setVariable(int type, String name, Object value) throws EventException {
    getEnv(type).setVariable(name, value);
  }
  public void clearVariable(int type, String name) throws EventException {
    getEnv(type).clearVariable(name);
  }
*/
  public void clearVariables(int scope, String name[]) throws EventException {
    for(int i=0; i<name.length; i++) {
      clearVariable(scope, name[i]);
    }
  }

  public void clearVariable(int scope, String name) throws EventException {
    getEnv(scope).clearVariable(name);
  }
  public void removeVariable(String name) throws EventException {
    Environment env = null;
    char c = name.charAt(0);
    if (c == 's' && name.startsWith("session.")) {
      env = sessionEnv;
      name = name.substring(8);
    } else if (c == 'a' && name.startsWith("application.")) {
      env = appEnv;
      name = name.substring(12);
    } else if (c == 'd' && name.startsWith("document.")) {
      env = docEnv;
      name = name.substring(9);
    } else if (c == 'd' && name.startsWith("dialog.")) {
      env = dialogEnv;
      name = name.substring(7);
    } else if (c == 'l' && name.startsWith("local.")) {
      env = dialogEnv;
      name = name.substring(6);
    } else {
      env = findEnvironment(name);
      if (env == null) {
        throw new EventException("error.semantic",
                    "variable not found " + name);
      }
    }
    env.removeVariable(name);
  }
  public void beginAnonymousScope() {
    currentEnv = new Environment(currentEnv);
  }
  public void endAnonymousScope() {
    currentEnv = currentEnv.getParent();
  }
  public void beginDialogScope() {
    dialogEnv = new Environment(docEnv);
    currentEnv = dialogEnv;
  }
```

```
public void beginDocumentScope() {
   docEnv = new Environment(appEnv);
   currentEnv = docEnv;
}
public void beginApplicationScope() {
   enterApplicationRoot();
   currentEnv = appEnv;
   currentEnv.removeBindings();
}
public void enterApplicationRoot() {
   docEnv = appEnv;
}
/**
 * Expression
 */
public Object evalExpression(Expr e) throws EventException {
   ExprImpl expr = (ExprImpl) e;
   String src = expr.getExpression();
   Object[] tokens = (Object[]) expr.getExpressionObject();
   if (tokens == null) {
      Vector v = new ExprLex(src).tokenize();
      tokens = new Object[v.size()];
      v.copyInto(tokens);
      expr.setExpressionObject(tokens);
   }
   return new Expression(interp,
                  new MyEnumeration(tokens),
                  src).evaluate();
}
public Object evalString(String str) throws EventException {
   Vector v = new ExprLex(str).tokenize();
   Object[] tokens = new Object[v.size()];
   v.copyInto(tokens);
   return new Expression(interp,
                  new MyEnumeration(tokens),
                  str).evaluate();
}
static private class MyEnumeration implements Enumeration {
   int i = 0;
   int max = 0;
   Object tokens[];
   MyEnumeration(Object a[]) {
      max = a.length;
      tokens = a;
   }
   public boolean hasMoreElements() {
      return i < max;
   }
   public Object nextElement() {
      return tokens[i++];
   }
}
private VXMLInterp interp = null;
private Environment sessionEnv = new Environment(null);
private Environment appEnv = new Environment(sessionEnv);
private Environment docEnv = new Environment(appEnv);
```

```
    private Environment dialogEnv = new Environment(docEnv);
    private Environment currentEnv = sessionEnv;
}
```

Document ID: 0.7.2.20

```
package com.genmagic.invui.vxml.property;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.VXMLException;
public class SearchPathPropertyListener implements PropertyChangeListener {
   public void propertyAdded(VXMLInterp ip, Object value) throws VXMLException {
      ip.getSpeechControl().setSearchPath(value.toString());
   }
   public void propertyRemoved(VXMLInterp ip) throws VXMLException {
      String original =
         ip.getCurrentProperties().getProperty("searchpath",
                                    null);
      if (original != null) {
         ip.getSpeechControl().setSearchPath(original);
      } else {
         // set to default
      }
   }
};
```

Document ID: 1

```
package com.genmagic.invui.vxml.interp;
public class SemanticException extends RuntimeException {
   public SemanticException(String str) {
      super(str);
   }
}
```

Document Ref. #

```
package com.genmagic.invui.vxml.prompt;
import java.util.Vector;
import java.util.Enumeration;
public class ShufflePrompts extends PromptSet {
  synchronized protected Vector construct() {
    Vector cache = new Vector();
    for (int i=0; i<10; i++) { // 10 patterns
      shuffle();
      generate(cache);
    }
    return cache;
  }
  private void shuffle() {
    int size = _prompts.size();
    for (int i = size * 3;
        i >=0;
        i--) {
      swap( (int) (((float)size) * rand.nextFloat()),
          (int) (((float)size) * rand.nextFloat()));
    }
  }
  private void swap(int i, int j) {
    Object ii = _prompts.elementAt(i);
    Object jj = _prompts.elementAt(j);
    _prompts.setElementAt(ii, j);
    _prompts.setElementAt(jj, i);
  }
  private void generate(Vector cache) {
    int size = _prompts.size();
    int base = 0;
    int six  = size % 6;
    int five = size % 5;
    int four = size % 4;
    int min = Math.min(six, five);
    int max = Math.max(six, five);
    if (size <= 8) {
      min = Math.min(min, four);
      max = Math.max(max, four);
    }
    if (min == 0) {
      if (min == six) {
        base = 6;
      } else if (min == five) {
        base = 5;
      } else if (min == four) {
        base = 4;
      }
    } else {
      if (max == six) {
        base = 5;
      } else if (max == five) {
        base = 4;
      } else if (max == four) {
        base = 3;
      }
    }
```

```
        int rem = size % base;
        int num = (size - rem) / base;

        for (int i=0; i<size; ) {
            int n = base;
            if (rem-->0) n++;
            StringBuffer buf = new StringBuffer();
            buf.append("<var name='session.shuffle' expr='-1'/>\n");
            buf.append("<if cond='0 > shuffle '>\n");
            buf.append("  <var name='application.shuffle' expr='0'/>\n");
            buf.append("<else/>\n");
            buf.append("  <assign name='application.shuffle' expr='application.shuffle + 1'/>\n");
            buf.append("</if>\n");
            buf.append("<var name='tmp' expr='application.shuffle % " + n + "'/>\n");
            String name = (String)_prompts.elementAt(i);
            buf.append("<if cond='tmp == 0'>\n");
            buf.append("  <audio src='builtin:" +
                    name + "'> " + name + " not found</audio>\n");
            i++;
            for (int j=1; j < n && i< size; j++, i++) {
                buf.append("<elseif cond='tmp == " + j + "'/>\n");
                name = (String)_prompts.elementAt(i);
                buf.append("  <audio src='builtin:" +
                        name + "'> " + name + " not found</audio>\n");
            }
            buf.append("</if>");
            cache.addElement(buf.toString());
        }
        if (cache.size()==0) {
            cache.addElement("");
        }
    }
    public static void main(String a[]) throws Exception {
        ShufflePrompts r = new ShufflePrompts();
        r.setAddPrompt("a.wav");
        r.setAddPrompt("b.wav");
        r.setAddPrompt("c.wav");
        r.setAddPrompt("d.wav");
        r.setAddPrompt("e.wav");
        r.setAddPrompt("f.wav");
        r.setAddPrompt("g.wav");
        r.setAddPrompt("h.wav");
        r.setAddPrompt("i.wav");
        r.setAddPrompt("j.wav");
        r.setAddPrompt("k.wav");
        r.setAddPrompt("l.wav");
        r.setAddPrompt("m.wav");
        r.setAddPrompt("n.wav");
        r.setAddPrompt("o.wav");
        r.setAddPrompt("p.wav");
        r.toVXML();
        System.out.println(r.toVXML());
    }
}
```

```
package com.genmagic.invui.vxml.interp;
public class SlotResult {
   public SlotResult(Object v, float c, String u) {
      value = v;
      confidence = c;
      utterance = u;
   }
   public Object value;
   public float confidence;
   public String utterance;
   public String toString() {
      return "Slot{value=" + value + ",conf=" + confidence + ",utt=" + utterance + "}";
   }
}
```

```
package com.genmagic.invui.vxml.interp;
import com.genmagic.invui.vxml.prompt.CompositePrompt;
public interface SpeechControl extends CompositePrompt {
  /*=========================================
   * PromptControl
   *=========================================*/
  public void backupPrompts();
  public void setCount(int c);
  public boolean checkCount(int c);
  public int getCount();
  public void setBargein(boolean b);
  public boolean getBargein();
  public void setTimeout(int timeout);
  public int getTimeout();
  public void clearTimeout();
  public void setReprompt(boolean b);
  public boolean getReprompt();
  /**
   * Speech State Control
   */
  public void enterState(String name) throws VXMLException;
  public void leaveState() throws VXMLException;
  // public Prompt[] getPrompts();
  /*
   * Plays prompts and waits until all play requests complete.
   */
  public void playPrompt() throws VXMLException ;
  /*
   * Plays prompts in async/sync mode.
   */
  public void playPrompt2(boolean async) throws VXMLException ;
  public void stopPlay(boolean wait_segment) throws VXMLException ;
  /**
   * Plays prompt and start recognition. It returns the result of the
   * recognition. If there is no speech, or no match with grammar,
   * it throws noinput, nomatch event respectively.
   */
  public RecResult playAndRecognition(Grammar gram, int noinput_timeout,
                                      int toomuch_timeout,
                                      int endspeech_timeout,
                                      int maxdigit) throws VXMLException;
  /**
   * Records user's input and return a VoiceXML variable associated with
   * recorded file.
   */
  public RecordVar record(int timeout,
                          int toomuch_timeout,
                          int endspeech_timeout) throws VXMLException;
  /**
   * Transfers the current call to given number.
   * @retrurn result.
   * @see VXMLConstants
   */
  public int transferCall(String number, byte[] uui, int timeout, int maxtime, boolean wait) throws VXMLException;
  /**
   * The termchar, which is ad-hoc way to terminate transfer call is
```

```
     * added for the moment.
     */
    public int transferCall(String number, byte[] uui, int timeout, int maxtime, boolean wait, String termChars) throws
VXMLException;
    /**
     * Accept the current incoming call.
     * @exception EventException with error.xxx if there is no
     *          incoming call which is not accepted.
     */
    public void acceptCall() throws VXMLException;
    /**
     * Drops all Calls
     */
    public void dropCalls();
    /**
     * Properties
     */
    public void setSearchPath(String sp) throws VXMLException ;
    /**
     * Set parameters
     */
    public void setParameter(String name, String value) throws VXMLException ;
    public void enableLog();
    /**
     * notifications
     */
}
```

```
package com.genmagic.invui.vxml.dom;
abstract public class Submit extends VXMLElement implements DocumentFetchable {
   public Submit() {
      super("submit");
   }
   /**
    * Get next
    */
   abstract public String getNext();
   /**
    * Get next
    * @since Vxml1.0
    */
   abstract public Expr getExpr();
   /**
    *
    */
   abstract public String[] getNameList();
   /**
    *
    */
   abstract public int getMethod();
   /**
    *
    */
   abstract public String getEnctype();
}
```

```
package com.genmagic.invui.vxml.tags;
import java.lang.Object;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.dom.Submit;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLException;
import com.genmagic.invui.vxml.interp.TagHandler;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.GotoNextException;
import com.genmagic.invui.vxml.interp.SpeechControl;
import com.genmagic.invui.vxml.prompt.CompositePrompt;
public class SubmitTag implements TagHandler {
   public void perform(VXMLInterp ip, Element self) throws VXMLException {
      Submit sm = (Submit)self;
      String next = sm.getNext();
      if (next == null) {
         Expr expr = sm.getExpr();
         if (expr != null) {
            next = ip.evalExpression(expr).toString();
         } else {
            throw new EventException("error.semantic",
                              "no next info in submit");
         }
      }
      // fetch hint
      // fetch audio
      SpeechControl speech = ip.getSpeechControl();
      CompositePrompt backup = null;
      String str = sm.getAttribute2("fetchAudioBreak");
      boolean breakFetchAudio = str != null ? Boolean.valueOf(str).booleanValue() : true;
      String src = sm.getFetchAudio();
      if(src != null)
      {
         backup = speech.getCompositePrompt();
         speech.clearPrompts();
         src = ip.loadAudio(src,
                     sm.getCaching(),
                     sm.getFetchTimeout());
         speech.appendAudio(src, "");
         speech.playPrompt2(true); // async play prompts
      }
      else
      {
         if (sm.hasChildNodes())
         {
            backup = speech.getCompositePrompt();
            speech.clearPrompts();
            ip.evalChildren(sm);
            speech.getCompositePrompt();
            speech.playPrompt2(true); // async play prompts
         }
      }
      try
      {
```

```
            ip.submit(next,
                sm.getNameList(),
                sm.getMethod(),
                sm.getEnctype(),
                sm.getCaching(),
                sm.getFetchTimeout());
          } catch(VXMLException ex) {
            throw ex;
          } finally {
            if(backup != null)
            {
              speech.stopPlay(!breakFetchAudio);
              speech.clearPrompts();
              // restore old prompt
              backup.appendTo(speech);
            }
          }
        } // perform
};
```

```
package com.genmagic.invui.vxml.interp;
import org.w3c.dom.Element;
/**
 * TagHandler interface is abstract parent for all tag implementation.
 */
public interface TagHandler {
   /**
    * Performs action specific to the tag it implements. It may throw
    * EventException if this tag or its child tag decided to throw
    * a VoiceXML event.
    * @param ip the interpreter object
    * @param element the dom element the interptert is currently visiting
    * @exception EventException if the tag or child node of this element
    *          thows an event.
    */
   public void perform(VXMLInterp ip, Element element) throws VXMLException;
}
```

Document ID: 5050292

```
package com.genmagic.invui.vxml.interp;
import org.omg.CORBA.UserException;
/*
import com.genmagic.mts.Session;
import com.genmagic.mts.MTSException;
*/
import com.genmagic.invui.vxml.idl.invui.*;
import com.genmagic.invui.vxml.idl.invui.DynamicGrammarPackage.*;
import com.genmagic.invui.vxml.idl.invui.DynamicGrammarPackage.GrammarData;
/*
 *
 */
public class TextGrammar extends DynamicGrammar {
   protected String src = null;
   private String key = null;
   private float _weight = -1.0f;
   public TextGrammar(String s) {
      src = s;
   }
   public TextGrammar(String s, float w) {
      src = s;
      _weight = w;
   }
   public String getName() {
      return DYNAMIC_TOP;
   }
   public String getUniqID() {
      return getKey();
   }
   public String getKey() {
      if (key == null) {
         key = genKey(src);
      }
      return key;
   }
   public float getWeight() {
      return _weight; // for the time being
   }
   public String getActiveGrammarNames() {
      throw new AssertException("should not be called");
   }
   public String getText() {
      return src;
   }
   public String[] getSlotNames() {
      return null;
   }
   public boolean equals(Object obj) {
      if (obj instanceof TextGrammar) {
         TextGrammar eg = (TextGrammar)obj;
         return src.equals(eg.src);
      }
      return false;
   }
   public int hashCode() {
      return src.hashCode();
```

```
    }
/*
    public void insertGrammar(Session sess) throws MTSException {
      sess.insertDynamicGrammar(getKey(), DYNAMIC_ROOT);
    }
    public void install(Session session) throws MTSException {
      String key = getKey();
      if (session.hasDynamicGrammar(key) == false) {
        session.createDynamicGrammar(key, src);
        int stat = session.waitDynamicGrammarCreation();
        if (stat != DG_Success) {
          throw new MTSException("Fail to create dynamic grammar :" +
                    DG_STATUS_STRING[stat]);
        }
      }
    }
*/
    public GrammarHandle getGrammarHandle(VUISession session) throws UserException, EventException {
      com.genmagic.invui.vxml.idl.invui.DynamicGrammar gram =
        new com.genmagic.invui.vxml.idl.invui.DynamicGrammar();
      GrammarData data = new GrammarData();
      data.key = getKey();
      data.grammar = getText();
      gram.text(data);
      try {
        return session.getGrammarHandle(gram);
      } catch (SpeechError ex) {
        throw new EventException("error.mts.speech",
                    "could not compile grammar : error_code="
                    + ex.error_code + ", source="+
                    data.grammar);
      }
    }
    public String toString() {
      return "Inline Text(" + key + "):" + src + ':' + super.toString();
    }
}
```

```
package com.genmagic.invui.vxml.dom;
abstract public class Throw extends VXMLElement {
   public Throw() {
      super("throw");
   }
   abstract public String getEventName();
}
```

Oracle Ex. 2010, p. 2010

```
package com.genmagic.invui.vxml.tags;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.dom.Throw;
import com.genmagic.invui.vxml.interp.TagHandler;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.VXMLException;
public class ThrowTag implements TagHandler {
   public void perform(VXMLInterp ip, Element self) throws VXMLException {
      Throw t = (Throw)self;
      throw new EventException(t.getEventName());
   }
}
```

Document Page 2 of 2

```
package com.genmagic.invui.vxml.dom;
import org.w3c.dom.Node;
import com.genmagic.invui.vxml.interp.EventException;
abstract public class Transfer extends FormItem {
   protected Transfer() {
      super("transfer");
   }
   abstract public String getDest();
   abstract public Expr getDestExpr() throws EventException;
   /**
    * Get the encoded UUI
    * @extension
    */
   abstract public String getUui();
   /**
    * @since Vxml1.0
    */
   abstract public int getConnectTimeout();
   /**
    * @since Vxml1.0
    */
   abstract public int getMaxTime();
   /**
    * @deprecated as of Vxml1.0
    */
   abstract public int getTimeout();
   /**
    * @deprecated as of Vxml1.0
    */
   abstract public int getDuration();
   abstract public boolean getBridge();
}
```

```
package com.genmagic.invui.vxml.dialogengine;
import com.genmagic.invui.vxml.idl.invui.Call;
import com.genmagic.invui.vxml.interp.VXMLException;
public interface TransferHandler {
   public TransferHandler clone(VXMLInterpCORBAImpl ip);
   public int transferCall(String number, byte[] uui, int timeout, String termChars) throws VXMLException;
   public int waitForTransferCallTermination(int maxtime) throws VXMLException;
   public void callDropped(Call call);
   public boolean callConnected(Call call);
   public void dropCalls();
   public void dtmf(char c);
}
```

Cisco Exhibit 2010 - Page 20

```
package com.genmagic.invui.vxml.dialogengine;
import com.genmagic.invui.vxml.idl.invui.Call;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLException;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.SpeechControl;
import com.genmagic.invui.vxml.interp.VXMLConstants;
public class TransferHandlerATTImpl implements TransferHandler, VXMLConstants {
  static final int CALLING = -1;
  int _state = TR_FAR_END_DISCONNECTED;
  VXMLInterpCORBAImpl _interp;
  StringBuffer _dtmfBuffer = new StringBuffer();
  TransferHandlerATTImpl() {
  }
  TransferHandlerATTImpl(VXMLInterpCORBAImpl ip) {
    _interp = ip;
  }
  public TransferHandler clone(VXMLInterpCORBAImpl ip) {
    return new TransferHandlerATTImpl(ip);
  }
  /**
   * UUI will be ignored.
   */
  public int transferCall(String number, byte[] uui, int timeout, String termChars) throws VXMLException {
    SpeechControl speech = _interp.getSpeechControl();
    speech.appendDTMF("*8");
    speech.appendDTMF(number);
    synchronized(_interp) {
      _state = CALLING;
      _dtmfBuffer.setLength(0);
      speech.playPrompt();
      if (timeout == 0) {
        timeout = Integer.MAX_VALUE;
      }
      long timeLeft = timeout;
      long until = timeLeft + System.currentTimeMillis();
      while (_state == CALLING &&
          _interp.isDropped() == false &&
          timeLeft >0) {
        try {
          _interp.wait(timeLeft);
        } catch (InterruptedException ex) {}
        timeLeft = until - System.currentTimeMillis();
      }
      if (_state == CALLING) {
        return TR_NOANSWER;
      } else if (_state == TR_CONNECTED) {
        speech.dropCalls();
        return TR_CONNECTED;
      } else {
        return _state;
      }
    }
  }
  public int waitForTransferCallTermination(int maxtime) throws VXMLException {
    throw new EventException("error.internal", "should not be called");
```

```
    }
    public void callDropped(Call call) {
      // nothing to do
    }
    public boolean callConnected(Call call) {
      return false;
    }
    public void dropCalls() {
      // cancel call!
    }
    static private final String DISCONNECTED = "**1";
    static private final String CANNOT_REDIRECT = "**5";
    static private final String CALL_TRANSFERED = "**6";
    static private final String CANNOT_RECOGNIZE = "**7";
    static private final String INVALID_COMMAND  = "**8";

    public void dtmf(char d) {
      synchronized(_interp) {
        if (_state == CALLING) {
          _dtmfBuffer.append(d);
          if (_dtmfBuffer.toString().length() < 2) {
            return;
          }
          String dtmf = _dtmfBuffer.toString();
          if (CALL_TRANSFERED.equals(dtmf)) {
            _state = TR_CONNECTED;
            _interp.notify();
          } else if (DISCONNECTED.equals(dtmf)) {
            _state = TR_NEAR_END_DISCONNECTED;
            _interp.notify();
          } else if (CANNOT_REDIRECT.equals(dtmf)) {
            _state = TR_BUSY;
            _interp.notify();
          } else if (CANNOT_RECOGNIZE.equals(dtmf)) {
            _state = TR_NOANSWER;
            _interp.notify();
          } else if (INVALID_COMMAND.equals(dtmf)) {
            _state = TR_ERROR;
            _interp.notify();
            // disconnecting
          } else {
            _state = TR_ERROR;
            _interp.notify();
          }
        }
      }
    }
}
```

```
package com.genmagic.invui.vxml.dialogengine;
import org.omg.CORBA.*;
import com.genmagic.invui.vxml.idl.invui.*;
import com.genmagic.invui.vxml.interp.VXMLException;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLConstants;
import com.genmagic.invui.vxml.interp.EventException;
final class TransferHandlerMTSImpl implements TransferHandler, VXMLConstants {
  /* for transfer */
  private static final int CALLING = -1;
  private VXMLInterpCORBAImpl _interp;
  private int _transferState = TR_FAR_END_DISCONNECTED;
  private Call _trans_call;
  private String _termChars = "%%%";
  private StringBuffer _currentTermChars = new StringBuffer();
  private long _previousDTMF = 0;
  public TransferHandlerMTSImpl() {
  }
  TransferHandlerMTSImpl(VXMLInterpCORBAImpl ip) {
    _interp = ip;
  }
  public TransferHandler clone(VXMLInterpCORBAImpl ip) {
    return new TransferHandlerMTSImpl(ip);
  }
  private void info(String msg) {
    _interp.info(msg);
  }
  private void diag(String msg) {
    _interp.diag(msg);
  }
  private void error(String msg) {
    _interp.error(msg);
  }
  public int transferCall(String number, byte[] uui, int timeout, String termChars) throws VXMLException {
    try {
      _termChars = termChars;
      _transferState = CALLING;
      Call call = _interp.getCall();
      _trans_call = call.session.placeCall(number, "123", uui, timeout);
      _interp.getDialogEngine().addSession(_trans_call, _interp);

      diag("waiting for connection :" + _trans_call.id);
      if (timeout == 0) {
        timeout = Integer.MAX_VALUE;
      }
      synchronized(_interp) {
        long timeLeft = timeout;
        long until = timeLeft + System.currentTimeMillis();
        while (_interp.isDropped() == false &&
            _transferState == CALLING &&
            timeLeft > 0) {
          try {
            _interp.wait(timeLeft);
          } catch (InterruptedException ex) { }
          timeLeft = until - System.currentTimeMillis();
        }
```

```
            if (_transferState != TR_CONNECTED) {
              try {
                info("cancelling transfer call");
                _trans_call.session.terminate();
              } catch (Exception ex) {}
              info("transfer call timeout");
              _interp.check();
              return TR_NOANSWER;
              // throw new Timeout();
            } else {
              _interp.check();
            }
          }
          diag("connecting calls : " + call.id + ", " + _trans_call.id);
          call.session.connectCalls(call, _trans_call);
          return TR_CONNECTED;
        } catch (UserBusy ex) {
          info("transfer call UserBusy : " + ex.getMessage());
          return TR_BUSY;
        } catch (NetworkBusy ex) {
          info("transfer call NetworkBusy : " + ex.getMessage());
          return TR_BUSY;
          // throw new EventException("busy", ex.getMessage());
        } catch (NoMoreChannel ex) {
          info("transfer call NoMoreChannel : " + ex.getMessage());
          // Should it be event?
          return TR_NETWORK_BUSY;
          // throw new EventException("error.nomorechannel", ex.getMessage());
        } catch (CallRejected ex) {
          info("transfer call CallRejected : " + ex.getMessage());
          return TR_NOANSWER;
        } catch (NetworkError ex) {
          info("transfer call NetworkError : " + ex.getMessage());
          return TR_NETWORK_BUSY;
        } catch (InvalidNumber ex) {
          info("transfer call InvalidNumber : " + ex.getMessage());
          return TR_NOANSWER;
        } catch (Timeout ex) {
          info("transfer call Timeout : " + ex.getMessage());
          return TR_NOANSWER;
          // throw new EventException("wrongnubmer", ex.getMessage());
        } catch (TelephonyError ex) {
          error("telephony error : " + ex.getMessage());
          throw new EventException("error.mts.telehony_error", ex.getMessage());
        } catch (UserException ex) {
          error("error occured in transfer: " + ex.getMessage());
          throw new EventException("error.mts.corba.user", ex.getMessage());
        } catch (SystemException ex) {
          error("system error occured in transfer: " + ex.getMessage());
          throw new EventException("error.mts.corba.system", ex.getMessage());
        }
      }
      public int waitForTransferCallTermination(int maxtime) throws VXMLException {
        synchronized(_interp) {
          if (maxtime == 0) {
```

```
            maxtime = Integer.MAX_VALUE;
        }
        long timeLeft = maxtime;
        long until = timeLeft + System.currentTimeMillis();

        while (_interp.isDropped() == false &&
               _transferState == TR_CONNECTED &&
               timeLeft > 0) {
            try {
                _interp.wait(timeLeft);
            } catch (InterruptedException ex) {}
            timeLeft = until - System.currentTimeMillis();
        }

        if (_transferState == TR_CONNECTED) {
            try {
                _trans_call.session.terminate();
            } catch (Exception ex) {}
            _transferState = TR_NETWORK_DISCONNECTED;
        }
        _interp.check();
    }
    return _transferState;
}
public void callDropped(Call droppedCall) {
    synchronized(_interp) {
        if (_trans_call.id == droppedCall.id) {
            info("transfer call dropped " + _trans_call.id);
            if (_transferState != TR_NEAR_END_DISCONNECTED) {
                _transferState = TR_FAR_END_DISCONNECTED;
            }
            _trans_call = null;
            _interp.notifyAll();
        }
    }
}
public boolean callConnected(Call connectedCall) {
    synchronized(_interp) {
        if (_transferState == CALLING &&
            _trans_call.id == connectedCall.id) {
            _transferState = TR_CONNECTED;
            _interp.notifyAll();
            return true;
        }
    }
    return false;
}
public void dropCalls() {
    Call tc = _trans_call;
    if (tc != null) {
        diag("dropping trombone call");
        try {
            tc.session.terminate();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
```

```
    }
  }
  public void dtmf(char c) {
    long current = System.currentTimeMillis();
    long diff = current - _previousDTMF;
    _previousDTMF = current;
    if (diff > 1000) {
      _currentTermChars.setLength(0);
    }
    _currentTermChars.append(c);
System.out.println("DTMF " + _termChars + " vs " +  _currentTermChars);
    if (_currentTermChars.toString().equals(_termChars)) {
      synchronized (_interp) {
        if (_transferState == TR_CONNECTED) {
System.out.println("disconnecting");
          _transferState = TR_NEAR_END_DISCONNECTED;
          try {
            _trans_call.session.terminate();
          } catch (Exception ex) {
            ex.printStackTrace();
            // ignore
          }
        } else {
System.out.println("cancelling");
          _transferState = TR_NEAR_END_DISCONNECTED;
          _interp.notifyAll();
        }
        // TERMINATE
        System.out.println("TERMINATE");
      }
    }
  }
}
```

```
package com.genmagic.invui.vxml.tags;
import java.lang.Object;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.dom.Transfer;
import com.genmagic.invui.vxml.dom.Filled;
import com.genmagic.invui.vxml.dom.Catch;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.interp.TagHandler;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventCounter;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.GotoNextItemException;
import com.genmagic.invui.vxml.interp.RuntimeEventException;
import com.genmagic.invui.vxml.interp.VXMLException;
import com.genmagic.invui.vxml.interp.VXMLConstants;
import com.genmagic.invui.vxml.interp.SpeechControl;
import com.genmagic.invui.vxml.interp.Util;
public class TransferTag implements TagHandler, VXMLConstants {
  public void perform(VXMLInterp ip, Element self) throws VXMLException {
    Transfer trans = (Transfer)self;
    String dest = trans.getDest();
    if (dest == null) {
      Expr expr = trans.getDestExpr();
      if (expr != null) {
        dest = ip.evalExpression(expr).toString();
      }
    }
    if (dest == null) {
      throw new EventException("error.semantic",
                    "no destination number in transfer");
    } else if (dest.startsWith("phone://") == false) {
      throw new EventException("error.semantic",
                    "only phone: is supported");
    }
    dest = dest.substring(8);
    boolean bridge = trans.getBridge();
    String varName = trans.getName();
    int timeout = trans.getConnectTimeout();
    if (timeout <0) {
      timeout =
        (int)ip.getCurrentProperties().getTimeProperty("connectimeout",
                                        30000);
    }
    int maxtime = trans.getMaxTime();
    if (maxtime <0) {
      maxtime =
        (int)ip.getCurrentProperties().getTimeProperty("maxtime",
                                        0);
    }
    String uuiStr = trans.getUui();
    byte[] uui = new byte[0];
    if (uuiStr != null) {
      try {
        uui = Util.decodeBASE64(uuiStr);
      } catch (java.io.IOException ex) {
        ip.error("error in decoding uui : uuiStr=" + uuiStr + ", err=" +
```

```
                ex.getMessage());
            }
        }
        ip.setVariable(trans.getName(), Boolean.TRUE);
        SpeechControl speech = ip.getSpeechControl();
        boolean connected = true;
        speech.playPrompt();
        String termChars = System.getProperty("jde.transfer.default_termchars", "%%%");
        // ## is just for AJ.
        int code = speech.transferCall(dest, uui, timeout, maxtime, bridge, termChars);
        Object value = null;
        switch (code) {
        case TR_FAR_END_DISCONNECTED:
            value = "far_end_disconnected";
            break;
        case TR_NEAR_END_DISCONNECTED:
            value = "near_end_disconnected";
            break;
        case TR_NETWORK_DISCONNECTED:
            value = "network_disconnected";
            break;
        case TR_CONNECTED:
            if (bridge == false) {
                throw new EventException("telephone.disconnect.transfer");
            } else {
                value = Boolean.TRUE;
                break;
            }
        case TR_BUSY:
            value = "busy";
            break;
        case TR_NETWORK_BUSY:
            value = "network_busy";
            break;
        case TR_NOANSWER:
            value = "noanswer";
            break;
        default:
            throw new EventException("error.internal", "unknown transfer result : " + code);
        }
        ip.setVariable(varName, value);
        Filled filled[] = trans.getParentForm().getFilled(new String[]{varName}, ip);
        if (filled != null) {
            ip.eval(filled);
        }
    }
}
```

```
package com.genmagic.invui.vxml.domimpl;
import com.genmagic.invui.vxml.interp.RuntimeEventException;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.Util;
import com.genmagic.invui.vxml.dom.Assign;
import com.genmagic.invui.vxml.dom.Expr;
class TXAssign extends Assign {
   Expr expr = null;
   String name = null;
   public Expr getExpr() throws EventException {
      if (expr == null) {
         expr = Util.getExprAttribute(this, "expr");
         if (expr == null) {
            throw new RuntimeEventException("error.semantic",
                                 "assign requires expr attribute :" +
                                 getName());
         }
      }
      return expr;
   }

   public String getName() {
      if (name == null) {
         // synchronized
         name = getAttribute2("name");
      }
      return name;
   }
}
```

```
package com.genmagic.invui.vxml.domimpl;
import com.genmagic.invui.vxml.interp.NotImplementedException;
import com.genmagic.invui.vxml.interp.Util;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.dom.Audio;
import com.genmagic.invui.vxml.dom.Expr;
class TXAudio extends Audio {
   private String src = NULL;
   private Expr exprsrc = null;
   private FetchInfo fetchInfo = null;
   public String getSrc() {
      if (src == NULL) {
         // synchronized
         src = getAttribute2("src");
      }
      return src;
   }

   public Expr getExprSrc() throws EventException {
      if (exprsrc == null) {
         exprsrc = Util.getExprAttribute(this, "exprsrc");
      }
      return exprsrc;
   }

   public String getMode() {
      throw new NotImplementedException("mode is deprecated in 1.0");
   }
   public int getCaching() {
      if (fetchInfo == null) {
         fetchInfo = new FetchInfo(this);
      }
      return fetchInfo.caching;
   }
   public int getFetchTimeout() {
      if (fetchInfo == null) {
         fetchInfo = new FetchInfo(this);
      }
      return fetchInfo.fetchTimeout;
   }
   public int getFetchHint() {
      if (fetchInfo == null) {
         fetchInfo = new FetchInfo(this);
      }
      return fetchInfo.fetchHint;
   }
}
```

```
package com.genmagic.invui.vxml.domimpl;
import org.w3c.dom.Node;
import com.genmagic.invui.vxml.dom.Block;
import com.genmagic.invui.vxml.dom.Dialog;
import com.genmagic.invui.vxml.dom.Catch;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.dom.Filled;
import com.genmagic.invui.vxml.dom.Property;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventHandlerSet;
import com.genmagic.invui.vxml.interp.EventCounter;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.VXMLDocumentImpl;
import com.genmagic.invui.vxml.interp.NotImplementedException;
class TXBlock extends Block {
   private FormItemInfo formItemInfo = null;
   private String varName = null;
   private String stateName = null;
   private EventHandlerSet set = null;
   private final void check() {
      if (formItemInfo == null) {
         formItemInfo = new FormItemInfo(this, "_block");
      }
   }
   public String getName() {
      check();
      return formItemInfo.name;
   }
   public String getStateName() {
      check();
      return formItemInfo.stateName;
   }
   public Expr getExpr() throws EventException {
      check();
      return formItemInfo.getExpr();
   }
   public Expr getCond() throws EventException {
      check();
      return formItemInfo.getCond();
   }
   public EventCounter getEventCounter() {
      check();
      return formItemInfo.getEventCounter();
   }
   public void setDefaultGuardVariable(VXMLInterp ip) throws EventException {
      check();
      formItemInfo.setDefaultGuardVariable(ip);
   }
   public Property[] getPropertyDecls() {
      check();
      return formItemInfo.propertyDecls;
   }
   public Catch getEventHandler(VXMLInterp ip, Exception ev, int count) throws EventException {
      if (set == null) {
         // synchronized
         set = new EventHandlerSet(this);
```

```
        }
        return set.getEventHandler(ip, ev, count);
    }
    public void getFilled(java.util.Vector v) {
        for (Node child = getFirstChild();
             child != null;
             child = child.getNextSibling()) {
            if (child instanceof Filled) {
                v.addElement(child);
            }
        }
    }
}
```

Document ID: 0.7.2.2.0.2.0

```
package com.genmagic.invui.vxml.domimpl;
import com.genmagic.invui.vxml.dom.Break;
import com.genmagic.invui.vxml.interp.RuntimeEventException;
import com.genmagic.invui.vxml.interp.Util;
import com.genmagic.invui.vxml.interp.NotImplementedException;
class TXBreak extends Break {
   static private final short UNDEF = -2;
   private short _size = UNDEF;
   private int _length = (int)UNDEF;
   public short getSize() {
      if (_size == UNDEF) {
         String sizeStr = getAttribute2("size");
         if (sizeStr == null) {
            _size = NONE;
         } else if ("small".equals(sizeStr)) {
            _size = SMALL;
         } else if ("medium".equals(sizeStr)) {
            _size = MEDIUM;
         } else if ("large".equals(sizeStr)) {
            _size = LARGE;
         } else {
            throw new RuntimeEventException("error.semantic",
                                    "illegal size of break:"+
                                    sizeStr);
         }
      }
      return _size;
   }
   public int getMsecs() {
      if (_length == (int)UNDEF) {
         String lengthStr = getAttribute2("msecs");
         if (lengthStr == null) {
            //these are default value
            switch (getSize()) {
            case SMALL:
               _length = 1000;
               break;
            case MEDIUM:
               _length = 2000;
               break;
            case LARGE:
               _length = 4000;
               break;
            default:
               throw new RuntimeEventException("error.semantic",
                                       "no size/length specified in break");
            }
         } else {
            _length = (int)Util.parseTime(lengthStr);
         }
      }
      return _length;
   }
   public int getLength() {
      throw new NotImplementedException("length is deprecated in 1.0");
   }
```

}

Document is the property of

```
package com.genmagic.invui.vxml.domimpl;
import com.genmagic.invui.vxml.dom.Catch;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.interp.Util;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.RuntimeEventException;
class TXCatch extends Catch {
   private String[] eventNames = null;
   private Expr cond = NULL_EXPR;
   private int count = -1;
   TXCatch() {}
   TXCatch(String name, String en) {
      super(name);
      eventNames = new String[1];
      eventNames[0] = en;
   }
   public String getEventName() {
      return getAttribute2("event");
/*
      if (eventName==null) {
         if (eventName == null) {
            throw new RuntimeEventException("error.semantic",
                                    "catch needs eventName");
         }
      }
      return eventName;
*/
   }
   public Expr getCond() throws EventException {
      if (cond == NULL_EXPR) {
         cond = Util.getExprAttribute(this, "cond");
      }
      return cond;
   }
   public int getCount() {
      if(count <0 ) {
         String countStr = getAttribute2("count");
         if (countStr == null) {
            count = 1;
         } else {
            try {
               count = Integer.parseInt(countStr);
            } catch (Exception ex) {
               throw new RuntimeEventException("error.semantic",
                                    "count should be integer");
            }
            if (count < 0) {
               throw new RuntimeEventException("error.semantic",
                                    "count should be positive");
            }
         }
      }
      return count;
   }
   public boolean match(VXMLInterp ip, String eventName) throws EventException {
```

```
        if (eventNames == null) {
            String thisEventName = getAttribute2("event");
            if (thisEventName == null) {
                throw new EventException("error.semantic",
                                "no event name specified in catch");
            }
            eventNames = Util.toArray(thisEventName);
            if (eventNames.length ==0 ) {
                throw new EventException("error.semantic",
                                "no event name specified in catch");
            }
        }
        // String thisEventName = getEventName();
        for (int i=0; i<eventNames.length; i++) {
            if (eventName.startsWith(eventNames[i])) {
                Expr cond = getCond();
                return (cond == null ? true :
                        ip.evalBooleanExpression(cond));
            }
        }
        return false;
    }
}
```

```
package com.genmagic.invui.vxml.domimpl;
import org.w3c.dom.Node;
import org.w3c.dom.CharacterData;
import com.genmagic.invui.vxml.dom.Choice;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.interp.RuntimeEventException;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.BuiltInGrammar;
import com.genmagic.invui.vxml.interp.TextGrammar;
import com.genmagic.invui.vxml.interp.Grammar;
import com.genmagic.invui.vxml.interp.GrammarAction;
import com.genmagic.invui.vxml.interp.NotImplementedException;
import com.genmagic.invui.vxml.interp.ExprImpl;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.RecResult;
/**
 *
 */
class TXChoice extends Choice {
    DocumentFetchInfo fetchInfo = null;
    String dtmf = NULL;
    LinkInfo linkInfo = null;
    public Expr getExpr() {
        if (linkInfo == null) {
            linkInfo = new LinkInfo(this);
        }
        return linkInfo.expr;
    }
    /**
     *
     */
    public String getNext() {
        if (linkInfo == null) {
            linkInfo = new LinkInfo(this);
        }
        return linkInfo.next;
    }
    /**
     *
     */
    public String getDtmf() {
        if (dtmf == NULL) {
            dtmf = getAttribute2("dtmf");
        }
        return dtmf;
    }

    /**
     * The child content
     */
    public String getContent() {
        StringBuffer buf = new StringBuffer();
        for (Node child = getFirstChild();
            child != null;
            child = child.getNextSibling()) {
```

```
        if (child instanceof CharacterData) { // text/pcdata
          buf.append(child.getNodeValue());
        }
      }
      return buf.toString();
    }
    public String getFetchAudio() {
      if (fetchInfo == null) {
        fetchInfo = new DocumentFetchInfo(this);
      }
      return fetchInfo.fetchAudio;
    }
    public int getCaching() {
      if (fetchInfo == null) {
        fetchInfo = new DocumentFetchInfo(this);
      }
      return fetchInfo.caching;
    }
    public int getFetchTimeout() {
      if (fetchInfo == null) {
        fetchInfo = new DocumentFetchInfo(this);
      }
      return fetchInfo.fetchTimeout;
    }
    public int getFetchHint() {
      if (fetchInfo == null) {
        fetchInfo = new DocumentFetchInfo(this);
      }
      return fetchInfo.fetchHint;
    }
    /**
     * REMIND: GMI Extension
     */
    public String getName() {
      throw new NotImplementedException("name is deprecated in 1.0");
    }
    /**
     * REMIND: GMI Extension
     */
    public String getSrc() {
      throw new NotImplementedException("src is deprecated in 1.0");
    }

    /**
     * REMIND: GMI Extension
     */
    public String getContentType() {
      throw new NotImplementedException("contenttype is deprecated in 1.0");
    }
    class MenuAction implements GrammarAction {
      Choice choice = null;
      MenuAction(Choice cs) {
        choice = cs;
      }
      public void performAction(Grammar g, RecResult r, Object cxt) throws Exception {
        VXMLInterp ip = (VXMLInterp)cxt;
```

```
        // REMIND : Nuance Specific
        String next = choice.getNext();
        if (next == null) {
          Expr expr = choice.getExpr();
          if (expr != null) {
            next = ip.evalExpression(expr).toString();
          } else {
            throw new EventException("error.semantic",
                                     "no next in choice");
          }
        }

        ip.gotoNext(next,
                    choice.getCaching(),
                    choice.getFetchTimeout());
      }
    }
    public Grammar getGrammar(VXMLInterp ip) throws EventException {
      com.genmagic.invui.vxml.interp.Grammar gram = null;
      // called only once
      for (Node child = getFirstChild();
           child != null;
           child = child.getNextSibling()) {
        if (child instanceof com.genmagic.invui.vxml.dom.Grammar) {
          gram = ((com.genmagic.invui.vxml.dom.Grammar)child).getGrammar(ip);
        }
      }
      if (gram == null) {
        gram = new TextGrammar(getContent());
      }
      gram.setAction(new MenuAction(this));
      return gram;
    }
}
```

```
package com.genmagic.invui.vxml.domimpl;
import org.w3c.dom.Node;
import com.genmagic.invui.vxml.interp.VXMLDocumentImpl;
import com.genmagic.invui.vxml.interp.NotImplementedException;
import com.genmagic.invui.vxml.dom.Clear;
class TXClear extends Clear {
   String[] names = null;
   private boolean updated = false;
   public String[] getNameList() {
      if (updated == false) {
         // synchronized
         updated = true;
         String name = getAttribute2("namelist");
         if (name != null) {
            names = VXMLDocumentImpl.getNames(name);
         }
      }
      return names;
   }
   public String[] getName() {
      throw new NotImplementedException("name is deprecated as of 1.0");
   }
/*
      public Form getCurrentForm() {
         if (form == null) {
            // synchronized

            for (Node e = getParentNode();
                 e != null;
                 e = e.getParentNode()) {
               if (e instanceof Form) {
                  form = (Form)e;
                  break;
               }
            }
         }
         return form;
      }
*/
}
```

```
package com.genmagic.invui.vxml.domimpl;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.VXMLInterp;
public class TXDebugCatch extends TXCatch {
   static boolean ignoreError = Boolean.getBoolean("jde.debugger.ignoreerror");
   static public void setIgnoreError(boolean b) {
      ignoreError = b;
   }
   public TXDebugCatch() {}
   public TXDebugCatch(String name, String en) {
      super(name, en);
   }
   public boolean match(VXMLInterp ip, String eventName) throws EventException {
      if (eventName.startsWith("error") && ignoreError) {
         return false;
      } else {
         return super.match(ip, eventName);
      }
   }
}
```

```
package com.genmagic.invui.vxml.domimpl;
import com.genmagic.invui.vxml.dom.Enumerate;
class TXEnumerate extends Enumerate {
   private String content = null;
   public String getStaticContent() {
      return content;
   }
   public void setStaticContent(String c) {
      content = c;
   }
}
```

```
package com.genmagic.invui.vxml.domimpl;
import com.genmagic.invui.vxml.interp.ExprImpl;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.VXMLDocumentImpl;
import com.genmagic.invui.vxml.interp.Util;
import com.genmagic.invui.vxml.dom.Exit;
import com.genmagic.invui.vxml.dom.Expr;
class TXExit extends Exit {
   private Expr expr = null;
   private String[] names = null;
   public Expr getExpr() throws EventException {
      if (expr == null) {
         expr = Util.getExprAttribute(this, "expr");
      }
      return expr;
   }
   public String[] getNameList() {
      if (names == null) {
         String name = getAttribute2("namelist");
         if (name != null) {
            names = VXMLDocumentImpl.getNames(name);
         } else {
            names = null;
         }
      }
      return names;
   }
}
```

```
package com.genmagic.invui.vxml.domimpl;
import java.util.Vector;
import org.w3c.dom.Node;
import com.genmagic.invui.vxml.interp.ExprImpl;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.EventHandlerSet;
import com.genmagic.invui.vxml.interp.EventCounter;
import com.genmagic.invui.vxml.interp.GrammarSet;
import com.genmagic.invui.vxml.interp.Grammar;
import com.genmagic.invui.vxml.interp.BuiltInGrammar;
import com.genmagic.invui.vxml.interp.RuntimeEventException;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLDocumentImpl;
import com.genmagic.invui.vxml.interp.NotImplementedException;
import com.genmagic.invui.vxml.interp.Util;
import com.genmagic.invui.vxml.dom.Catch;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.dom.Field;
import com.genmagic.invui.vxml.dom.Form;
import com.genmagic.invui.vxml.dom.Filled;
import com.genmagic.invui.vxml.dom.Link;
import com.genmagic.invui.vxml.dom.Dialog;
import com.genmagic.invui.vxml.dom.Property;
final class TXField extends Field {
   private FormItemInfo formItemInfo = null;
   private Node[] prompts = null;
   private String slotName = null;
   private GrammarSet grammars = null;
   private boolean modal = false;
   private EventHandlerSet set = null;
   private Form form = null;
   private Grammar main_gram = null;
   private final void check() {
      if (formItemInfo == null) {
         formItemInfo = new FormItemInfo(this, "_field");
      }
   }
   public String getName() {
      check();
      return formItemInfo.name;
   }
   public Expr getExpr() throws EventException {
      return formItemInfo.getExpr();
   }
   public Expr getCond() throws EventException {
      check();
      return formItemInfo.getCond();
   }
   public String getStateName() {
      check();
      return formItemInfo.stateName;
   }
   public EventCounter getEventCounter() {
      check();
      return formItemInfo.getEventCounter();
   }
```

```
public void setDefaultGuardVariable(VXMLInterp ip) throws EventException {
  check();
  formItemInfo.setDefaultGuardVariable(ip);
}
public Property[] getPropertyDecls() {
  check();
  return formItemInfo.propertyDecls;
}
private void update() {
  modal = Util.getBooleanAttribute(this, "modal", false);
  slotName = getAttribute2("slot");
  if (slotName == null) {
    if (main_gram != null) {
      String[] slots = main_gram.getSlotNames();
      if (slots != null) {
        if (slots.length != 1) {
          throw new RuntimeEventException("error.semantic",
                                          "field grammar should have one slot");
        }
        slotName = slots[0];
      }
    }
    if (slotName == null) {
      slotName = getName();
    }
  }
}
public String getType() {
  return getAttribute2("type");
}
public boolean getModal() {
  if (slotName == null) {
    update();
  }
  return modal;
}
public String getSlot() {
  if (slotName == null) {
    update();
  }
  return slotName;
}
public int getTimeout() {
  throw new NotImplementedException("timeout is deprecated in 1.0");
}
public int getToomuchTimeout() {
  throw new NotImplementedException("toomuchTimeout is deprecated in 1.0");
}
public int getEndspeechTimeout() {
  throw new NotImplementedException("endspeechTimeout is deprecated in 1.0");
}
public Catch getEventHandler(VXMLInterp ip, Exception ev, int count) throws EventException {
  if (set == null) {
    // synchronized
    set = new EventHandlerSet(this);
  }
```

```
    return set.getEventHandler(ip, ev, count);
  }
  /**
   * Returns prompt related children only.
   * prompt, audio, PCDATA, value, enumerate
   */
  public Node[] getPromptElements() {
    if (prompts == null) {
      prompts = Util.getPromptElements(this);
    }
    return prompts;
  }
  public void getFilled(Vector v) {
    for (Node child = getFirstChild();
         child != null;
         child = child.getNextSibling()) {
      if (child instanceof Filled) {
        v.addElement(child);
      }
    }
  }
  public GrammarSet getActiveGrammars(VXMLInterp ip) throws EventException {
    if (grammars == null) {
      grammars = Util.getActiveGrammars(ip, this);
    }
    return grammars;
  }
  public Grammar[] getGrammars(VXMLInterp ip) throws EventException {
    // this is called only once.
    if (main_gram == null) {
      String type= getType();
      if (type != null) {
        main_gram = BuiltInGrammar.getGrammarByType(type);
        if (main_gram == null) {
          throw new RuntimeEventException("error.semantic",
                                          "built in grammar for " + type +
                                          " not found");
        }
      }
    }
    Vector v = new Vector();
    for (Node child = getFirstChild();
         child != null;
         child = child.getNextSibling()) {
      if (child instanceof com.genmagic.invui.vxml.dom.Grammar) {
        com.genmagic.invui.vxml.dom.Grammar g =
          (com.genmagic.invui.vxml.dom.Grammar)child;
        main_gram = g.getGrammar(ip);
      } else if (child instanceof Link) {
        Grammar[] g = ((Link)child).getLinkGrammars(ip);
        for (int i=0; i<g.length; i++) {
          v.addElement(g[i]);
        }
      }
    }
    if (main_gram == null) {
```

```
        return new Grammar[0];
/*
        throw new RuntimeEventException("error.semantic",
                                        "no grammar for field");
*/
      }
      v.addElement(main_gram);
      Grammar[] g = new Grammar[v.size()];
      v.copyInto(g);
      return g;
    }
    public Form getParentForm() {
      if (form == null) {
        form = super.getParentForm();
      }
      return form;
    }
}
```

```
package com.genmagic.invui.vxml.domimpl;
import org.w3c.dom.Node;
import com.genmagic.invui.vxml.interp.RuntimeEventException;
import com.genmagic.invui.vxml.interp.VXMLDocumentImpl;
import com.genmagic.invui.vxml.dom.Filled;
import com.genmagic.invui.vxml.dom.FormItem;
import com.genmagic.invui.vxml.dom.Form;
class TXFilled extends Filled {
   String[] names = null;
   private boolean mode_read = false;
   private boolean mode = ANY;
   public boolean getMode() {
      if (mode_read==false) {
         // synchronzied
         mode_read = true;
         String m = getAttribute2("mode");
         if ("any".equals(m)) {
            mode = ANY;
         } else if ("all".equals(m)) {
            mode = ALL;
         } else if (m != null) {
            throw new RuntimeEventException("error.semantic",
                                    "invalid mode value :" + m);
         }
      }
      return mode;
   }
   public String[] getNameList() {
      if (names == null) {
         Node node = getParentNode();
         if (node instanceof FormItem) {
            // ignore if parent is Field
            names = new String[]{((FormItem)node).getName()};
/*
         } else if (node instanceof Record) {
            // REMIND: GMI Extension
            names = new String[]{((Record)node).getName()};
*/
         } else {
            String name = getAttribute2("namelist");
            if (name != null) {
               names = VXMLDocumentImpl.getNames(name);
            } else if (node instanceof Form) {
               Form form = (Form)node;
               names = form.getAllGuardVariables();
            } else {
               throw new RuntimeEventException("error.semantic",
                                       "Parent is not Form");
            }
         }
      }
      return names;
   }
}
```

```
package com.genmagic.invui.vxml.domimpl;
import java.util.Vector;
import org.w3c.dom.Node;
import org.w3c.dom.Document;
import com.genmagic.invui.vxml.interp.RuntimeEventException;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.EventHandlerSet;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLConstants;
import com.genmagic.invui.vxml.interp.VXMLDocumentImpl;
import com.genmagic.invui.vxml.interp.Grammar;
import com.genmagic.invui.vxml.interp.GrammarSet;
import com.genmagic.invui.vxml.interp.GrammarAction;
import com.genmagic.invui.vxml.interp.GotoNextItemException;
import com.genmagic.invui.vxml.interp.GotoNextException;
import com.genmagic.invui.vxml.interp.Util;
import com.genmagic.invui.vxml.interp.NotImplementedException;
import com.genmagic.invui.vxml.interp.RecResult;
import com.genmagic.invui.vxml.dom.Catch;
import com.genmagic.invui.vxml.dom.Form;
import com.genmagic.invui.vxml.dom.FormItem;
import com.genmagic.invui.vxml.dom.Field;
import com.genmagic.invui.vxml.dom.Block;
import com.genmagic.invui.vxml.dom.Transfer;
import com.genmagic.invui.vxml.dom.Initial;
import com.genmagic.invui.vxml.dom.Filled;
import com.genmagic.invui.vxml.dom.Link;
import com.genmagic.invui.vxml.dom.Dialog;
import com.genmagic.invui.vxml.dom.Record;
import com.genmagic.invui.vxml.dom.Property;
import com.genmagic.invui.vxml.dom.Var;
import com.genmagic.invui.vxml.dom.VXMLElement;
import com.genmagic.invui.vxml.dom.Expr;
class TXForm extends Form implements VXMLConstants {
  static class FormGrammarAction implements GrammarAction, VXMLConstants {
    Form form = null;
    FormGrammarAction(Form f, String id) {
      form = f;
    }
    public void performAction(Grammar grammar, RecResult r, Object cxt) throws Exception {
      VXMLInterp ip = (VXMLInterp)cxt;
      String[] varNames =
        Util.copyVariablesFromRecResult(ip,
                                        form.getFormItems(),
                                        r);
      Filled[] filled = form.getFilled(varNames, ip);
      if (filled != null) {
        ip.eval(filled);
      }
      throw new GotoNextItemException();
    }
  }
  static class DocumentGrammarAction implements GrammarAction {
    Dialog dialog = null;
    DocumentGrammarAction(Dialog d) {
      dialog = d;
```

```
    }
    public void performAction(Grammar grammar, RecResult r, Object cxt) throws Exception {
      VXMLInterp ip = (VXMLInterp)cxt;
      Document doc = dialog.getOwnerDocument();
      if (doc == ip.getAppContext().getDocument()) {
        throw new GotoNextException(dialog,
                                    r,
                                    IN_DOCUMENT);
      } else if (doc == ip.getAppContext().getApplicationRootDocument()) {
        ip.getAppContext().setApplicationRootAsCurrent();
        throw new GotoNextException(dialog,
                                    r,
                                    IN_APPLICATION_ROOT);
      }
    }
  }
  private String id = null;
  private FormItem[] items = null;
  private VXMLElement[] decls = null;
  private Filled[] filled = null;
  private Initial initial = null;
  private Grammar[] form_grammars = null;
  private Grammar[] document_grammars = null;
  private EventHandlerSet set = null;
  private int scope = LOCAL_GRAMMAR;
  public String getID() {
    if (id == null) {
      // synchronized
      id = getAttribute2("id");
      if (id == null) {
        id = VXMLDocumentImpl.genVarName("_form");
      }
      String scopeStr = getAttribute2("scope");
      if ("document".equals(scopeStr)) {
        scope = DOCUMENT_GRAMMAR;
      }
    }
    return id;
  }
  public int getScope() {
    if (id == null) {
      getID();
    }
    return scope;
  }
  public int getTimeout() {
    throw new NotImplementedException("timeout is deprecated as of 1.0");
  }
  public String getCost() {
    throw new NotImplementedException("cost is deprecated as of 1.0");
  }
  public boolean getBargein() {
    throw new NotImplementedException("bargein is deprecated as of 1.0");
  }
  public Var[] getVarDecls() {
    throw new NotImplementedException("getVarDecls is deprecated as of 1.0");
```

```
}
public VXMLElement[] getDecls() {
  if (decls == null) {
    Vector v = new Vector();
    for(Node child = getFirstChild();
      child != null;
      child = child.getNextSibling()) {
      if (child instanceof Var ||
        child instanceof FormItem ||
        child instanceof Property) {
        v.addElement(child);
      }
    }
    decls = new VXMLElement[v.size()];
    v.copyInto(decls);
  }
  return decls;
}
public FormItem[] getFormItems() {
  if (items == null) {
    Vector v = new Vector();
    for (Node child = getFirstChild();
      child != null;
      child = child.getNextSibling()) {
      if (child instanceof Field ||
        child instanceof Block ||
        child instanceof Transfer ||
        child instanceof Record ||
        child instanceof com.genmagic.invui.vxml.dom.Object) {
        v.addElement(child);
      } else if (child instanceof Initial) {
        initial = (Initial)initial;
        v.addElement(child);
      }
    }
    items = new FormItem[v.size()];
    v.copyInto(items);
  }
  return items;
}
public Catch getEventHandler(VXMLInterp ip, Exception ev, int count) throws EventException {
  if (set == null) {
    // synchronized
    set = new EventHandlerSet(this);
  }
  return set.getEventHandler(ip, ev, count);
}
public Initial getInitial() {
  if (items == null) {
    getFormItems();
  }
  return initial;
}
public void getFormGrammars(VXMLInterp ip, GrammarSet set) throws EventException {
  if (form_grammars == null) {
    Vector v = new Vector();
```

```
    GrammarAction action = new FormGrammarAction(this, getID());
    for (Node child = getFirstChild();
        child != null;
        child = child.getNextSibling()) {
      if (child instanceof com.genmagic.invui.vxml.dom.Grammar) {
        com.genmagic.invui.vxml.dom.Grammar dom_gram =
          (com.genmagic.invui.vxml.dom.Grammar)child;
        /*
         * DOCUMENT GRAMMAR SHOULD BE ACTIVE ANYWAY.
         * IF THIS IS PRESENT, THE GRAMMAR OBTAINED BY THE METHOD BELOW WILL
         * NOT BE REGISTERED. SEE Util#addGrammars
         if (dom_gram.getScope() == DOCUMENT_GRAMMAR) {
         continue;
         }
         */
        Grammar g = dom_gram.getGrammar(ip);
        g.setAction(action);
        v.addElement(g);
      } else if (child instanceof Link) {
        Grammar[] g = ((Link)child).getLinkGrammars(ip);
        for (int i=0; i<g.length; i++) {
          v.addElement(g[i]);
        }
      }
    }
    form_grammars = new Grammar[v.size()];
    v.copyInto(form_grammars);
  }
  set.addGrammars(form_grammars);
  // return form_grammars;
}
public Grammar[] getDocumentGrammars(VXMLInterp ip) throws EventException {
  if (document_grammars == null) {
    Vector v = new Vector();
    GrammarAction action = new DocumentGrammarAction(this);
    for (Node child = getFirstChild();
        child != null;
        child = child.getNextSibling()) {
      if (child instanceof com.genmagic.invui.vxml.dom.Grammar) {
        com.genmagic.invui.vxml.dom.Grammar dom_gram =
          (com.genmagic.invui.vxml.dom.Grammar)child;

        if (dom_gram.getScope() != DOCUMENT_GRAMMAR) {
          continue;
        }
        Grammar g = dom_gram.getGrammar(ip);
        g.setAction(action);
        v.addElement(g);
      }
    }
    document_grammars = new Grammar[v.size()];
    v.copyInto(document_grammars);
  }
  return document_grammars;
}
public Filled[] getFilled() {
```

```
    if (filled == null) {
      // synchronized;
      Vector v = new Vector();
      for (Node child = getFirstChild();
          child != null;
          child = child.getNextSibling()) {
        if (child instanceof Filled) {
          v.addElement(child);
        } else if (child instanceof FormItem) {
          ((FormItem)child).getFilled(v);
        }
        /* // REMIND: GMI Extension
          if (child instanceof TXRecord) {
            ((TXRecord)child).getFilled(v);
          }
        */
      }
      filled = new Filled[v.size()];
      v.copyInto(filled);
    }
    return filled;
  }
  public FormItem getNextFormItem(VXMLInterp ip) throws EventException {
    FormItem[] items = getFormItems();
    boolean init_visited = false;
    for (int i=0; i<items.length; i++) {
      FormItem item = items[i];
      Expr cond = item.getCond();
      String name = item.getName();
      if (ip.hasDefinedVariable(VXMLConstants.LOCAL, name) == false &&
        (cond == null ||
         ip.evalBooleanExpression(cond) == true)) {
        // NOTE: Initial is NOT the first formitem. If there
        // is a block before initial, that block should be
        // executed first.
        // when it first visited prompt enabled form items,
        // looks up initial element.
        if ( (item instanceof Field ||
              item instanceof Record) /* and transcribe */ &&
             init_visited == false) {
          init_visited = true;

          Initial init = getInitial();
          if (init == item) {
            return init;
          } else if (init != null) {
            cond = init.getCond();
            if (ip.hasDefinedVariable(VXMLConstants.LOCAL, init.getName())
              == false &&
              (cond == null ||
               ip.evalBooleanExpression(cond) == true)) {
              return init;
            }
          }
        }
        return item;
```

```
            }
        }
        return null;
    }
}
```

Document ID: 0.7.2.20

```
package com.genmagic.invui.vxml.domimpl;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.RuntimeEventException;
import com.genmagic.invui.vxml.interp.NotImplementedException;
import com.genmagic.invui.vxml.interp.VXMLConstants;
import com.genmagic.invui.vxml.interp.Util;
import com.genmagic.invui.vxml.dom.Goto;
import com.genmagic.invui.vxml.dom.Expr;
class TXGoto extends Goto implements VXMLConstants {
   private LinkInfo linkInfo = null;
   private String nextItem = NULL;
   private Expr exprItem = null;
   private Object exit = null; // see ExitTag.java
   private DocumentFetchInfo fetchInfo = null;
   public String getNextItem() {
      if (nextItem == NULL) {
         nextItem = getAttribute2("nextitem");
      }
      return nextItem;
   }
   public Expr getExprItem() throws EventException {
      if (exprItem == null) {
         exprItem = Util.getExprAttribute(this, "expritem");
      }
      return exprItem;
   }
   public String getNext() {
      if (linkInfo == null) {
         linkInfo = new LinkInfo(this);
      }
      return linkInfo.next;
   }
   public Expr getExpr() {
      if (linkInfo == null) {
         linkInfo = new LinkInfo(this);
      }
      return linkInfo.expr;
   }
   public String[] getSubmit() {
      throw new NotImplementedException("submit is deprecated in 1.0");
   }
   public int getMethod() {
      throw new NotImplementedException("submit is deprecated in 1.0");
   }
   public String getEnctype() {
      throw new NotImplementedException("enctype is deprecated in 1.0");
   }
   public int getCaching() {
      if (fetchInfo == null) {
         fetchInfo = new DocumentFetchInfo(this);
      }
      return fetchInfo.caching;
   }
   public int getFetchTimeout() {
      if (fetchInfo == null) {
         fetchInfo = new DocumentFetchInfo(this);
```

```
        }
        return fetchInfo.fetchTimeout;
    }
    public int getFetchHint() {
        if (fetchInfo == null) {
            fetchInfo = new DocumentFetchInfo(this);
        }
        return fetchInfo.fetchHint;
    }
    public String getFetchAudio() {
        if (fetchInfo == null) {
            fetchInfo = new DocumentFetchInfo(this);
        }
        return fetchInfo.fetchAudio;
    }
}
```

Document Ref: 1000000

```
package com.genmagic.invui.vxml.domimpl;
import org.w3c.dom.Node;
import org.w3c.dom.CharacterData;
import com.genmagic.invui.vxml.dom.VXMLDocument;
import com.genmagic.invui.vxml.dom.Form;
import com.genmagic.invui.vxml.interp.TextGrammar;
import com.genmagic.invui.vxml.interp.Grammar;
import com.genmagic.invui.vxml.interp.BuiltInGrammar;
import com.genmagic.invui.vxml.interp.RuntimeEventException;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.VXMLConstants;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.Util;
import com.genmagic.invui.vxml.interp.NotImplementedException;
class TXGrammar extends com.genmagic.invui.vxml.dom.Grammar implements VXMLConstants {
   private String src = null;
   private int scope = LOCAL_GRAMMAR;
   private float weight = -1.0f;
   private String type = null;
   private Grammar g = null;
   private FetchInfo fetchInfo = null;
   private boolean updated = false;
   public String getSrc() {
      if (updated == false) {
         update();
      }
      return src;
   }
   public int getScope() {
      if (updated == false) {
         update();
      }
      return scope;
   }
   public String getType() {
      if (updated == false) {
         update();
      }
      return type;
   }
   synchronized private void update() {
      if (updated == false) {
         updated = true;
         String scopeStr = getAttribute2("scope");
         if (scopeStr == null) {
            // do not use getParentForm();
            Node parent = getParentNode();
            if (parent instanceof Form) {
               scope = ((Form)parent).getScope();
            }
         } else if ("document".equals(scopeStr)) {
            scope = DOCUMENT_GRAMMAR;
         }
         type = getAttribute2("type");
         src = getAttribute2("src");
         if (src != null) {
```

```
        String srcURL =
          ((VXMLDocument)getOwnerDocument()).getSourceURI();
        // resolve if it's relative path.
        if (src.indexOf(':')<0) {
          try {
            src = Util.resolve(srcURL, src);
          } catch (Exception ex) {
            ex.printStackTrace();
            throw new RuntimeEventException("error.semantic",
                                "malformed url " + srcURL + " + " + src);
          }
        }
        // get grammar name if it's builtin
        if (src.startsWith("builtin:")) {
          // needs to be improved
          src = src.substring(8);
          if (src.startsWith("//")) {
            src = src.substring(2);
          } else if (src.startsWith("/")) {
            src = src.substring(1);
          }
        }

        String weightStr = getAttribute2("weight");
        if (weightStr != null) {
          try {
            weight = Float.parseFloat(weightStr);
          } catch (NumberFormatException ex) {
            throw new RuntimeEventException("error.semantic",
                                "wrong format in weight");
          }
        }
      }
    }
  }
  public Grammar getGrammar(VXMLInterp ip) throws EventException {
    if (updated == false) {
      update();
    }
    if (src != null) {
      if (src.indexOf(':')<0) {
        // needs to be improved
        return new BuiltInGrammar(src, getWeight());
      } else {
        String text = ip.loadGrammar(src,
                          getCaching(),
                          getFetchTimeout());
        // TYPE CHECK
        return new TextGrammar(text, getWeight());
      }
    } else {
      StringBuffer b = new StringBuffer();
      for (Node node = getFirstChild();
          node != null;
          node = node.getNextSibling()) {
        if (node instanceof CharacterData) {
```

```
            b.append(node.getNodeValue());
          }
        }
        if (type != null &&
          type.startsWith("text/")==false) {
          throw new RuntimeEventException("error.semantic",
                                "text/<subtype> is expected");
        }
        return new TextGrammar(b.toString(), getWeight());
      }
    }
    private float getWeight() {
      if (updated == false) {
        update();
      }
      return weight;
    }
    public int getCaching() {
      if (fetchInfo == null) {
        fetchInfo = new FetchInfo(this);
      }
      return fetchInfo.caching;
    }
    public int getFetchTimeout() {
      if (fetchInfo == null) {
        fetchInfo = new FetchInfo(this);
      }
      return fetchInfo.fetchTimeout;
    }
    public int getFetchHint() {
      if (fetchInfo == null) {
        fetchInfo = new FetchInfo(this);
      }
      return fetchInfo.fetchHint;
    }
  }
```

```
package com.genmagic.invui.vxml.domimpl;
import java.util.Vector;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import com.genmagic.invui.vxml.interp.RuntimeEventException;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.ExprImpl;
import com.genmagic.invui.vxml.dom.If;
import com.genmagic.invui.vxml.dom.Expr;
//import com.genmagic.invui.vxml.dom.If.IfBlock;
class TXIf extends If {
  IfBlock[] blocks = null;
  static class IfBlockImpl implements If.IfBlock {
    String exprStr = null;
    Expr expr = null;
    Node[] block = null;
    public IfBlockImpl(String e, Node b[]) {
      exprStr = e;
      block = b;
    }
    public Expr getExpr() throws EventException {
      if (expr == null && exprStr != null) {
        expr = new ExprImpl(exprStr);
      }
      return expr;
    }
    public Node[] getBlock() {
      return block;
    }
  }
  public IfBlock[] getIfBlocks() {
    if (blocks == null) {
      update();
    }
    return blocks;
  }
  private void update() {
    Vector b = new Vector();
    Vector v = new Vector();
    String exprStr = getAttribute2("cond");
    if (exprStr == null) {
      throw new RuntimeEventException("error.semantic",
                                      "if tag needs cond attribute");
    }
    for (Node node = getFirstChild();
         node != null;
         node = node.getNextSibling()) {
      if (node instanceof Element) {
        if ("else".equals(node.getNodeName())) {
          Node block[] = new Node[v.size()];
          v.copyInto(block);
          b.addElement(new IfBlockImpl(exprStr, block));
          v.setSize(0);
          exprStr = null;
          continue;
```

```
            } else if ("elseif".equals(node.getNodeName())) {
               Node block[] = new Node[v.size()];
               v.copyInto(block);
               b.addElement(new IfBlockImpl(exprStr, block));
               v.setSize(0);
               NamedNodeMap attrs = node.getAttributes();
               exprStr = null;
               if (attrs != null) {
                  Node cond = attrs.getNamedItem("cond");
                  if (cond != null) {
                     exprStr = cond.getNodeValue();
                  }
               }
               if (exprStr == null) {
                  throw new RuntimeEventException("error.semantic",
                                    "elseif needs cond attribute");
               }
               continue;
            }
         }
         v.addElement(node);
      }
      if (v.size()>0) {
         Node block[] = new Node[v.size()];
         v.copyInto(block);
         b.addElement(new IfBlockImpl(exprStr, block));
      }
      blocks = new IfBlock[b.size()];
      b.copyInto(blocks);
   }
}
```

```
package com.genmagic.invui.vxml.domimpl;
import org.w3c.dom.Node;
import java.util.Vector;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.EventHandlerSet;
import com.genmagic.invui.vxml.interp.EventCounter;
import com.genmagic.invui.vxml.interp.Grammar;
import com.genmagic.invui.vxml.interp.GrammarSet;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLDocumentImpl;
import com.genmagic.invui.vxml.interp.Util;
import com.genmagic.invui.vxml.interp.NotImplementedException;
import com.genmagic.invui.vxml.dom.Catch;
import com.genmagic.invui.vxml.dom.Dialog;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.dom.Filled;
import com.genmagic.invui.vxml.dom.Link;
import com.genmagic.invui.vxml.dom.Initial;
import com.genmagic.invui.vxml.dom.Property;
class TXInitial extends Initial {
   private FormItemInfo formItemInfo = null;
   private Node[] prompts = null;
   private GrammarSet grammars = null;
   private EventHandlerSet set = null;
   private final void check() {
      if (formItemInfo == null) {
         formItemInfo = new FormItemInfo(this, "_initial");
      }
   }
   public String getName() {
      check();
      return formItemInfo.name;
   }
   public Expr getExpr() throws EventException {
      check();
      return formItemInfo.getExpr();
   }
   public Expr getCond() throws EventException {
      check();
      return formItemInfo.getCond();
   }
   public String getStateName() {
      check();
      return formItemInfo.stateName;
   }
   public EventCounter getEventCounter() {
      check();
      return formItemInfo.getEventCounter();
   }
   public void setDefaultGuardVariable(VXMLInterp ip) throws EventException {
      check();
      formItemInfo.setDefaultGuardVariable(ip);
   }
   public Property[] getPropertyDecls() {
      check();
      return formItemInfo.propertyDecls;
```

```
}
public void getFilled(java.util.Vector v) {
   for (Node child = getFirstChild();
        child != null;
        child = child.getNextSibling()) {
      if (child instanceof Filled) {
         v.addElement(child);
      }
   }
}
/**
 * Returns prompt related children only.
 * prompt, audio, PCDATA, value, enumerate
 */
public Node[] getPromptElements() {
   if (prompts == null) {
      prompts = Util.getPromptElements(this);
   }
   return prompts;
}

public GrammarSet getActiveGrammars(VXMLInterp ip) throws EventException {
   if (grammars == null) {
      grammars = Util.getActiveGrammars(ip, this);
   }
   return grammars;
}
public Grammar[] getGrammars(VXMLInterp ip) throws EventException {
   // this is called only once.
   Vector v = new Vector();
   for (Node child = getFirstChild();
        child != null;
        child = child.getNextSibling()) {
      if (child instanceof Link) {
         Grammar[] g = ((Link)child).getLinkGrammars(ip);
         for (int i=0; i<g.length; i++) {
            v.addElement(g[i]);
         }
      }
   }
   Grammar[] g = new Grammar[v.size()];
   v.copyInto(g);
   return g;
}
public Catch getEventHandler(VXMLInterp ip, Exception ev, int count) throws EventException {
   if (set == null) {
      // synchronized
      set = new EventHandlerSet(this);
   }
   return set.getEventHandler(ip, ev, count);
}
public int getTimeout() {
   throw new NotImplementedException("timeout is deprecated as of 1.0");
}
public int getToomuchTimeout() {
   throw new NotImplementedException("timeout is deprecated as of 1.0");
```

```
    }
    public int getEndspeechTimeout() {
        throw new NotImplementedException("timeout is deprecated as of 1.0");
    }
}
```

Document Page - Inventor Corporation

```
package com.genmagic.invui.vxml.domimpl;
import java.util.Vector;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.interp.AppContext;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.RuntimeEventException;
import com.genmagic.invui.vxml.interp.GrammarAction;
import com.genmagic.invui.vxml.interp.Grammar;
import com.genmagic.invui.vxml.interp.VXMLConstants;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.NotImplementedException;
import com.genmagic.invui.vxml.interp.RecResult;
import com.genmagic.invui.vxml.dom.Link;
import com.genmagic.invui.vxml.dom.VXMLDocument;
import com.genmagic.invui.vxml.dom.Expr;
class TXLink extends Link implements VXMLConstants {
  static class LinkNextAction implements GrammarAction {
    String srcURL = null;
    Link link = null;
    LinkNextAction(String s, Link l) {
      srcURL = s;
      link = l;
    }
    public void performAction(Grammar g, RecResult r, Object cxt) throws Exception {
      VXMLInterp ip = (VXMLInterp)cxt;
      AppContext app = ip.getAppContext();
      try {
        if (app.isInSameApplicationRoot(srcURL)) {
          // do rollback if failed!
          app.setApplicationRootAsCurrent();
        }
      } catch (java.net.MalformedURLException ex) {
        ex.printStackTrace();
        // should not happen
        throw new EventException("error.fatal");
      }
      String next = link.getNext();
      if (next == null) {
        Expr expr = link.getExpr();
        if (expr != null) {
          next = ip.evalExpression(expr).toString();
        } else {
          throw new EventException("error.semantic",
                                   "link requires next, expr or event attribute");
        }
      }
      //REMIND: when to evaluate? currently when recognized.
      ip.gotoNext(next,
            link.getCaching(),
            link.getFetchTimeout());
    }
  }
  static class LinkEventAction implements GrammarAction {
    String event = null;
    LinkEventAction(String e) {
```

```
      event = e;
    }
    public void performAction(Grammar g, RecResult r, Object cxt) throws Exception {
      throw new EventException(event);
    }
  }
  Grammar[] grammars = null;
  String eventName = NULL;
  String dtmf = NULL;
  LinkInfo linkInfo = null;
  DocumentFetchInfo fetchInfo = null;
  public String getNext() {
    if (linkInfo == null) {
      linkInfo = new LinkInfo(this);
    }
    return linkInfo.next;
  }
  public Expr getExpr() throws EventException {
    if (linkInfo == null) {
      linkInfo = new LinkInfo(this);
    }
    return linkInfo.getExpr();
  }
  public String getDtmf() {
    if (dtmf == NULL) {
      dtmf = getAttribute2("dtmf");
    }
    return dtmf;
  }
  public String getEventName() {
    if (eventName == NULL) {
      eventName = getAttribute2("event");
    }
    return eventName;
  }
  public Grammar[] getLinkGrammars(VXMLInterp ip) throws EventException {
    if (grammars != null) {
      return grammars;
    }
    String e = getEventName();
    GrammarAction action = null;
    if (e != null) {
      action = new LinkEventAction(e);
    } else {
      String srcURL =
        ((VXMLDocument)getOwnerDocument()).getSourceURI();
      action = new LinkNextAction(srcURL, this);
    }
    Vector v = new Vector();
    for (Node child = getFirstChild();
      child != null;
      child = child.getNextSibling()) {
      if (child instanceof Element==false) {
        continue;
      }
      if (child instanceof com.genmagic.invui.vxml.dom.Grammar==false) { // if not???
```

```
            throw new RuntimeEventException("error.semantic",
                                "children of link should be grammars " + child.getNodeName());
          }
          Grammar g =
            ((com.genmagic.invui.vxml.dom.Grammar)child).getGrammar(ip);
          g.setAction(action);
          v.addElement(g);
        }
        grammars = new Grammar[v.size()];
        v.copyInto(grammars);
        return grammars;
      }
      public String getFetchAudio() {
        if (fetchInfo == null) {
          fetchInfo = new DocumentFetchInfo(this);
        }
        return fetchInfo.fetchAudio;
      }
      public int getCaching() {
        if (fetchInfo == null) {
          fetchInfo = new DocumentFetchInfo(this);
        }
        return fetchInfo.caching;
      }
      public int getFetchTimeout() {
        if (fetchInfo == null) {
          fetchInfo = new DocumentFetchInfo(this);
        }
        return fetchInfo.fetchTimeout;
      }
      public int getFetchHint() {
        if (fetchInfo == null) {
          fetchInfo = new DocumentFetchInfo(this);
        }
        return fetchInfo.fetchHint;
      }
    }
```

```
package com.genmagic.invui.vxml.domimpl;
import java.util.Vector;
import org.w3c.dom.Node;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.EventHandlerSet;
import com.genmagic.invui.vxml.interp.Grammar;
import com.genmagic.invui.vxml.interp.GrammarSet;
import com.genmagic.invui.vxml.interp.VXMLConstants;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLDocumentImpl;
import com.genmagic.invui.vxml.interp.NotImplementedException;
import com.genmagic.invui.vxml.interp.Util;
import com.genmagic.invui.vxml.dom.Menu;
import com.genmagic.invui.vxml.dom.Choice;
import com.genmagic.invui.vxml.dom.Catch;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.dom.Property;
import com.genmagic.invui.vxml.dom.VXMLElement;
import com.genmagic.invui.vxml.dom.Var;
class TXMenu extends Menu implements VXMLConstants {
  private String id = null;
  private VXMLElement[] decls = null;
  private Node[] prompts = null;
  private Choice[] choices = null;
  private Grammar grammar[] = null;
  private GrammarSet grammars = null;
  private EventHandlerSet set = null;
  private int scope = LOCAL_GRAMMAR;
  private boolean dtmf = true;
  public String getID() {
    if (id == null) {
      // synchronized
      id = getAttribute2("id");
      if (id == null) {
        id = VXMLDocumentImpl.genVarName("_menu");
      }
      String scopeStr = getAttribute2("scope");
      if ("document".equals(scopeStr)) {
        scope = DOCUMENT_GRAMMAR;
      }
      String dtmfStr = getAttribute2("dtmf");
      dtmf = dtmfStr != null ?
        Boolean.valueOf(dtmfStr).booleanValue() : true;
    }
    return id;
  }
  public int getScope() {
    if (id == null) {
      getID();
    }
    return scope;
  }
  public boolean getDtmf() {
    if (id == null) {
      getID();
    }
```

```
    return dtmf;
  }
  public boolean getBargein() {
    throw new NotImplementedException("bargein is deprecated as of 1.0");
  }
  public Node[] getPromptElements() {
    if (prompts == null) {
      prompts = Util.getPromptElements(this);
    }
    return prompts;
  }
  /**
   * Choice elements in this form
   */
  public Choice[] getChoices() {
    if (choices == null) {
      Vector v = new Vector();
      int i = 1;
      for (Node child = getFirstChild();
          child != null;
          child = child.getNextSibling()) {
        if (child instanceof Choice) {
          Choice c = (Choice)child;
          if (c.getDtmf() == null) {
            c.setAttribute("dtmf",
                    String.valueOf(v.size()+1));
          }
          v.addElement(child);
        }
      }
      choices = new Choice[v.size()];
      v.copyInto(choices);
    }
    return choices;
  }
  public VXMLElement[] getDecls() {
    if (decls == null) {
      Vector v = new Vector();
      for(Node child = getFirstChild();
        child != null;
        child = child.getNextSibling()) {
        if (child instanceof Var ||
          child instanceof Property) {
          v.addElement(child);
        }
      }
      decls = new VXMLElement[v.size()];
      v.copyInto(decls);
    }
    return decls;
  }
  /**
   * Grammar
   */
  public void getGrammar(VXMLInterp ip, GrammarSet set) throws EventException {
    updateGrammar(ip);
```

```
        set.addGrammars(grammar);
    }
    public Grammar[] getDocumentGrammars(VXMLInterp ip) throws EventException {
        if (getScope() == DOCUMENT) {
            updateGrammar(ip);
            return grammar;
        } else {
            return new Grammar[0];
        }
    }
    private void updateGrammar(VXMLInterp ip) throws EventException {
        if (grammar == null) {
            Choice[] choices = getChoices();
            Vector v = new Vector();
            for (int i=0; i<choices.length; i++) {
                Choice c = choices[i];
                v.addElement(choices[i].getGrammar(ip));
            }
            grammar = new com.genmagic.invui.vxml.interp.Grammar[v.size()];
            v.copyInto(grammar);
        }
    }
    public GrammarSet getActiveGrammars(VXMLInterp ip) throws EventException {
        if (grammars == null) {
            grammars = Util.getActiveGrammars(ip, this);
        }
        return grammars;
    }
    public Catch getEventHandler(VXMLInterp ip, Exception ev, int count) throws EventException {
        if (set == null) {
            // synchronized
            set = new EventHandlerSet(this);
        }
        return set.getEventHandler(ip, ev, count);
    }
    public int getTimeout() {
        throw new NotImplementedException("timeout is deprecated as of 1.0");
    }
    public String getCost() {
        throw new NotImplementedException("cost is deprecated as of 1.0");
    }
}
```

```
package com.genmagic.invui.vxml.domimpl;
import java.util.Vector;
import org.w3c.dom.Node;
import com.genmagic.invui.vxml.interp.AssertException;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.EventCounter;
import com.genmagic.invui.vxml.interp.GrammarAction;
import com.genmagic.invui.vxml.interp.Grammar;
import com.genmagic.invui.vxml.interp.GrammarSet;
import com.genmagic.invui.vxml.interp.VXMLConstants;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLDocumentImpl;
import com.genmagic.invui.vxml.interp.Util;
import com.genmagic.invui.vxml.interp.NotImplementedException;
import com.genmagic.invui.vxml.dom.Object;
import com.genmagic.invui.vxml.dom.Catch;
import com.genmagic.invui.vxml.dom.Dialog;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.dom.Filled;
import com.genmagic.invui.vxml.dom.Property;
import com.genmagic.invui.vxml.dom.Param;
class TXObject extends com.genmagic.invui.vxml.dom.Object {
    private final static Param[] NULL_PARAM = new Param[0];
    private FormItemInfo formItemInfo = null;
    private DocumentFetchInfo fetchInfo = null;
    private Param[] params = NULL_PARAM;
    private final void check() {
        if (formItemInfo == null) {
            formItemInfo = new FormItemInfo(this, "_object");
        }
    }
    public String getName() {
        check();
        return formItemInfo.name;
    }
    public Expr getExpr() throws EventException {
        check();
        return formItemInfo.getExpr();
    }
    public Expr getCond() throws EventException {
        check();
        return formItemInfo.getCond();
    }
    public EventCounter getEventCounter() {
        check();
        return formItemInfo.getEventCounter();
    }
    public void setDefaultGuardVariable(VXMLInterp ip) throws EventException {
        check();
        formItemInfo.setDefaultGuardVariable(ip);
    }
    public Property[] getPropertyDecls() {
        check();
        return formItemInfo.propertyDecls;
    }
    public Param[] getParams() {
```

```
    if (params == NULL_PARAM) {
        Vector v = new Vector();
        for (Node child = getFirstChild();
             child != null;
             child = child.getNextSibling()) {
            if (child instanceof Param) {
                v.addElement(child);
            }
        }
        params = new Param[v.size()];
        v.copyInto(params);
    }
    return params;
}
public void getFilled(Vector v) {
    for (Node child = getFirstChild();
         child != null;
         child = child.getNextSibling()) {
        if (child instanceof Filled) {
            v.addElement(child);
        }
    }
}
public String getFetchAudio() {
    if (fetchInfo == null) {
        fetchInfo = new DocumentFetchInfo(this);
    }
    return fetchInfo.fetchAudio;
}
public int getCaching() {
    if (fetchInfo == null) {
        fetchInfo = new DocumentFetchInfo(this);
    }
    return fetchInfo.caching;
}
public int getFetchTimeout() {
    if (fetchInfo == null) {
        fetchInfo = new DocumentFetchInfo(this);
    }
    return fetchInfo.fetchTimeout;
}
public int getFetchHint() {
    if (fetchInfo == null) {
        fetchInfo = new DocumentFetchInfo(this);
    }
    return fetchInfo.fetchHint;
}
public String getClassid() {
    return getAttribute2("classid");
}
public String getCodebase() {
    return getAttribute2("codebase");
}
public String getCodetype() {
    return getAttribute2("codetype");
}
```

```
    public String getData() {
      return getAttribute2("data");
    }
    public String getType() {
      return getAttribute2("type");
    }
    public String getArchive() {
      return getAttribute2("archive");
    }
    public String getStateName() {
      throw new AssertException("Object tag have no state");
    }
    public Catch getEventHandler(VXMLInterp ip, Exception ev, int count) throws EventException {
      Dialog d = (Dialog)getParentNode();
      return d.getEventHandler(ip, ev, count);
    }
    public String getSrc() {
      throw new NotImplementedException("expect is deprecated as of 1.0");
    }
    public String[] getSubmit() {
      throw new NotImplementedException("expect is deprecated as of 1.0");
    }
    public String[] getExpect() {
      throw new NotImplementedException("expect is deprecated as of 1.0");
    }
}
```

```
package com.genmagic.invui.vxml.domimpl;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.ExprImpl;
import com.genmagic.invui.vxml.interp.RuntimeEventException;
import com.genmagic.invui.vxml.interp.Util;
import com.genmagic.invui.vxml.dom.Param;
import com.genmagic.invui.vxml.dom.Expr;
class TXParam extends Param {
   Expr expr = null;
   String name = null;
   String value = NULL;
   public Expr getExpr() throws EventException {
      if (expr == null) {
         expr = Util.getExprAttribute(this, "expr");
      }
      return expr;
   }
   public String getValue() {
      if (value == NULL) {
         value = getAttribute2("value");
      }
      return value;
   }
   public String getValueType() {
      return getAttribute2("valuetype");
   }
   public String getType() {
      return getAttribute2("type");
   }
   public String getName() {
      if (name == null) {
         // synchronized
         name = getAttribute2("name");
         if (name == null) {
            throw new RuntimeEventException("error.semantic",
                                    "no name in var");
         }
      }
      return name;
   }
}
```

```
package com.genmagic.invui.vxml.domimpl;
import com.genmagic.invui.vxml.interp.RuntimeEventException;
import com.genmagic.invui.vxml.interp.Util;
import com.genmagic.invui.vxml.interp.ExprImpl;
import com.genmagic.invui.vxml.interp.NotImplementedException;
import com.genmagic.invui.vxml.interp.VXMLConstants;
import com.genmagic.invui.vxml.dom.Prompt;
import com.genmagic.invui.vxml.dom.Dialog;
import com.genmagic.invui.vxml.dom.Expr;
class TXPrompt extends Prompt implements VXMLConstants {
   private boolean updated = false;
   private Expr cond = null;
   private int timeout = -1;
   private int bargein = BARGEIN_NONE;
   private int count = 1;
   public int getTimeout() {
      if (updated==false) {
         update();
      }
      return timeout;
   }
   public int getBargein() {
      if (updated==false) {
         update();
      }
      return bargein;
   }
   public int getCount() {
      if (updated==false) {
         update();
      }
      return count;
   }
   public Expr getCond() {
      if (updated==false) {
         update();
      }
      return cond;
   }
   synchronized private void update() {
      if (updated == false) {
         updated = true;
         timeout = (int)Util.getTimeAttribute(this,
                                    "timeout",
                                    -1);

         String bargeinStr = getAttribute2("bargein");

         if (bargeinStr != null) {
            if ("true".equals(bargeinStr)) {
               bargein = BARGEIN_ON;
            } else if ("false".equals(bargeinStr)) {
               bargein = BARGEIN_OFF;
            } else {
               throw new RuntimeEventException("error.semantic",
                                    "invalid bargein format:"+
```

```
                                    bargeinStr);
                }
            }
            try {
                String countStr = getAttribute2("count");
                if (countStr != null) {
                    count = Integer.parseInt(countStr);
                }
                cond = Util.getExprAttribute(this, "cond");
            } catch (Exception ex) {
                throw new RuntimeEventException("error.semantic",
                                    "prompt " + ex.getMessage());
            }
        }
    }
}
```

Oracle Exhibit 1005 - Page 1

```
package com.genmagic.invui.vxml.domimpl;
import com.genmagic.invui.vxml.interp.Util;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.VXMLConstants;
import com.genmagic.invui.vxml.dom.Property;
import com.genmagic.invui.vxml.dom.Expr;
class TXProperty extends Property implements VXMLConstants {
   String name = null;
   String value = NULL;
   Expr expr = null;
   public String getName() {
      if (name == null) {
         name = getAttribute2("name");
      }
      return name;
   }
   public String getValue() {
      if (value == NULL) {
         value = getAttribute2("value");
      }
      return value;
   }
   public Expr getExpr() throws EventException {
      if (expr == null) {
         expr = Util.getExprAttribute(this, "expr");
      }
      return expr;
   }
}
```

```
package com.genmagic.invui.vxml.domimpl;
import java.util.Vector;
import org.w3c.dom.Node;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.EventHandlerSet;
import com.genmagic.invui.vxml.interp.EventCounter;
import com.genmagic.invui.vxml.interp.RuntimeEventException;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.Util;
import com.genmagic.invui.vxml.interp.NotImplementedException;
import com.genmagic.invui.vxml.dom.Catch;
import com.genmagic.invui.vxml.dom.Record;
import com.genmagic.invui.vxml.dom.Dialog;
import com.genmagic.invui.vxml.dom.Filled;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.dom.Property;
class TXRecord extends Record {
    private Node[] prompts = null;
    private EventHandlerSet set = null;
    private FormItemInfo formItemInfo = null;
    private String type = null;
    private int finalSilence = -2;
    private int maxtime;
    String dtmfTerm;
    private boolean beep;
    private boolean modal;
    private void check() {
        if (formItemInfo == null) {
            formItemInfo = new FormItemInfo(this, "_record");
        }
    }
    public Expr getExpr() throws EventException {
        check();
        return formItemInfo.getExpr();
    }
    public Expr getCond() throws EventException {
        check();
        return formItemInfo.getCond();
    }
    public String getName() {
        check();
        if (formItemInfo.name == null) {
            throw new RuntimeEventException("error.semantic",
                                "no name attribute in record");
        }
        return formItemInfo.name;
    }
    public String getStateName() {
        check();
        return formItemInfo.stateName;
    }
    public EventCounter getEventCounter() {
        check();
        return formItemInfo.getEventCounter();
    }
    public void setDefaultGuardVariable(VXMLInterp ip) throws EventException {
```

```
    check();
    formItemInfo.setDefaultGuardVariable(ip);
  }
  public Property[] getPropertyDecls() {
    check();
    return formItemInfo.propertyDecls;
  }
  public String getType() {
    if (finalSilence == -2) {
      update();
    }
    return type;
  }
  public int getMaxTime() {
    if (finalSilence == -2) {
      update();
    }
    return maxtime;
  }
  public int getFinalSilence() {
    if (finalSilence == -2) {
      update();
    }
    return finalSilence;
  }
  public String getDTMFTerm() {
    if (finalSilence == -2) {
      update();
    }
    return dtmfTerm;
  }
  public boolean getBeep() {
    if (finalSilence == -2) {
      update();
    }
    return beep;
  }
  public boolean getModal() {
    if (finalSilence == -2) {
      update();
    }
    return modal;
  }
  private void update() {
    type = getAttribute2("type");
    dtmfTerm = getAttribute2("dtmfterm");
    maxtime = Util.getTimeAttribute(this, "maxtime", -1);
    finalSilence = Util.getTimeAttribute(this, "finalsilence", -1);
    beep = Util.getBooleanAttribute(this, "beep", false);
    modal = Util.getBooleanAttribute(this, "modal", false);
  }
  public Node[] getPromptElements() {
    if (prompts == null) {
      prompts = Util.getPromptElements(this);
    }
    return prompts;
```

```
    }
    public Catch getEventHandler(VXMLInterp ip, Exception ev, int count) throws EventException {
        if (set == null) {
            // synchronized
            set = new EventHandlerSet(this);
        }
        return set.getEventHandler(ip, ev, count);
    }
    public void getFilled(Vector v) {
        for (Node child = getFirstChild();
             child != null;
             child = child.getNextSibling()) {
            if (child instanceof Filled) {
                v.addElement(child);
            }
        }
    }
    public int getMaxLength() {
        throw new NotImplementedException("maxlength is deprecated in 1.0");
    }
}
```

Document Number: 245127-001

```
package com.genmagic.invui.vxml.domimpl;
class TXReprompt extends com.genmagic.invui.vxml.dom.Reprompt {
}
```

```
package com.genmagic.invui.vxml.domimpl;
class TXSayas extends com.genmagic.invui.vxml.dom.Sayas {
   String klass = null;
   public String get_Class() {
      if (klass == null) {
         klass = getAttribute2("class");
      }
      return klass;
   }
}
```

```
package com.genmagic.invui.vxml.domimpl;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.RuntimeEventException;
import com.genmagic.invui.vxml.interp.NotImplementedException;
import com.genmagic.invui.vxml.interp.VXMLConstants;
import com.genmagic.invui.vxml.interp.Util;
import com.genmagic.invui.vxml.dom.Submit;
import com.genmagic.invui.vxml.dom.Expr;
class TXSubmit extends Submit implements VXMLConstants {
   static final String[] emptyNameList = new String[0];
   private LinkInfo linkInfo = null;
   private Object exit = null; // see ExitTag.java
   private DocumentFetchInfo fetchInfo = null;
   private int method = GET;
   private String enctype = null;
   private String[] namelist = null;
   public String getNext() {
      if (linkInfo == null) {
         linkInfo = new LinkInfo(this);
      }
      return linkInfo.next;
   }
   public Expr getExpr() {
      if (linkInfo == null) {
         linkInfo = new LinkInfo(this);
      }
      return linkInfo.expr;
   }
   public String[] getNameList() {
      if (namelist == null) {
         update();
      }
      return namelist;
   }
   public int getMethod() {
      if (namelist == null) {
         update();
      }
      return method;
   }
   public String getEnctype() {
      if (namelist == null) {
         update();
      }
      return enctype;
   }
   public int getCaching() {
      if (fetchInfo == null) {
         fetchInfo = new DocumentFetchInfo(this);
      }
      return fetchInfo.caching;
   }
   public int getFetchTimeout() {
      if (fetchInfo == null) {
         fetchInfo = new DocumentFetchInfo(this);
      }
```

```
        return fetchInfo.fetchTimeout;
    }
    public int getFetchHint() {
        if (fetchInfo == null) {
            fetchInfo = new DocumentFetchInfo(this);
        }
        return fetchInfo.fetchHint;
    }
    public String getFetchAudio() {
        if (fetchInfo == null) {
            fetchInfo = new DocumentFetchInfo(this);
        }
        return fetchInfo.fetchAudio;
    }
    private void update() {
        String names = getAttribute2("namelist");
        if (names != null) {
            namelist = Util.toArray(names);
        } else {
            namelist = emptyNameList;
        }

        String methodStr = getAttribute2("method");
        if (methodStr == null) {
        } else if ("post".equals(methodStr)) {
            method = POST;
        } else if ("get".equals(methodStr) == false) {
            throw new RuntimeEventException("error.semantic",
                                    "method needs to be get or post");
        }
        enctype = getAttribute2("enctype");
    }
}
```

```
package com.genmagic.invui.vxml.domimpl;
class TXThrow extends com.genmagic.invui.vxml.dom.Throw {
   String event = null;
   public String getEventName() {
      if (event == null) {
         event = getAttribute2("event");
      }
      return event;
   }
}
```

Document Page

```
package com.genmagic.invui.vxml.domimpl;
import org.w3c.dom.Node;
import com.genmagic.invui.vxml.interp.AssertException;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.EventHandlerSet;
import com.genmagic.invui.vxml.interp.EventCounter;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.Util;
import com.genmagic.invui.vxml.interp.VXMLDocumentImpl;
import com.genmagic.invui.vxml.interp.NotImplementedException;
import com.genmagic.invui.vxml.dom.Transfer;
import com.genmagic.invui.vxml.dom.Catch;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.dom.Filled;
import com.genmagic.invui.vxml.dom.Property;
class TXTransfer extends Transfer {
    private FormItemInfo formItemInfo = null;
    private EventHandlerSet set = null;
    private Expr destExpr = null;
    private int maxtime = -2;
    private int timeout = -2;
    public String getDest() {
        return getAttribute2("dest");
    }
    private final void check() {
        if (formItemInfo == null) {
            formItemInfo = new FormItemInfo(this, "_transfer");
        }
    }
    public String getName() {
        check();
        return formItemInfo.name;
    }
    public String getStateName() {
        throw new AssertException("transfer tag have no state");
    }
    public Expr getExpr() throws EventException {
        check();
        return formItemInfo.getExpr();
    }
    public String getUui() {
        return getAttribute2("uui");
    }
    public Expr getCond() throws EventException {
        check();
        return formItemInfo.getCond();
    }
    public EventCounter getEventCounter() {
        check();
        return formItemInfo.getEventCounter();
    }
    public void setDefaultGuardVariable(VXMLInterp ip) throws EventException {
        check();
        formItemInfo.setDefaultGuardVariable(ip);
    }
    public Property[] getPropertyDecls() {
```

```
    check();
    return formItemInfo.propertyDecls;
  }
  public void getFilled(java.util.Vector v) {
    for (Node child = getFirstChild();
         child != null;
         child = child.getNextSibling()) {
      if (child instanceof Filled) {
        v.addElement(child);
      }
    }
  }
  public Expr getDestExpr() throws EventException {
    if (destExpr == null) {
      destExpr = Util.getExprAttribute(this, "destexpr");
    }
    return destExpr;
  }
  public int getConnectTimeout() {
    if (timeout == -2) {
      timeout = Util.getTimeAttribute(this, "connecttimeout", -1);
    }
    return timeout;
  }
  public int getMaxTime() {
    if (maxtime == -2) {
      maxtime = Util.getTimeAttribute(this, "maxtime", -1);
    }
    return maxtime;
  }
  public int getTimeout() {
    throw new NotImplementedException("timeout is deprecated as of 1.0");
  }
  public int getDuration() {
    throw new NotImplementedException("duration is not implemented JDE1.0");
  }
  public boolean getBridge() {
    return Util.getBooleanAttribute(this, "bridge", true);
  }
  public Catch getEventHandler(VXMLInterp ip, Exception ev, int count) throws EventException {
    if (set == null) {
      // synchronized
      set = new EventHandlerSet(this);
    }
    return set.getEventHandler(ip, ev, count);
  }
}
```

```
package com.genmagic.invui.vxml.domimpl;
import com.genmagic.invui.vxml.interp.RuntimeEventException;
import com.genmagic.invui.vxml.interp.NotImplementedException;
import com.genmagic.invui.vxml.interp.Util;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.dom.Value;
import com.genmagic.invui.vxml.dom.Expr;
class TXValue extends Value {
   Expr expr = null;
   String klass = null;
   String mode = NULL;
   String recsrc = NULL;
   public String getName() {
      throw new NotImplementedException("name is deprecated as of Vxml1.0");
   }
   public Expr getExpr() throws EventException {
      if (expr == null) {
         expr = Util.getExprAttribute(this, "expr");
      }
      return expr;
   }
   public String getMode() {
      if (mode == NULL) {
         mode = getAttribute2("mode");
      }
      return mode;
   }
   public String getRecsrc() {
      if (recsrc == NULL) {
         recsrc = getAttribute2("recsrc");
      }
      return recsrc;
   }
   public String get_Class() {
      if (klass == null) {
         klass = getAttribute2("class");
      }
      return klass;
   }
}
```

```
package com.genmagic.invui.vxml.domimpl;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.ExprImpl;
import com.genmagic.invui.vxml.interp.RuntimeEventException;
import com.genmagic.invui.vxml.interp.Util;
import com.genmagic.invui.vxml.dom.Var;
import com.genmagic.invui.vxml.dom.Expr;
class TXVar extends Var {
   Expr expr = null;
   String name = null;
   public Expr getExpr() throws EventException {
      if (expr == null) {
         expr = Util.getExprAttribute(this, "expr");
      }
      return expr;
   }
   public String getName() {
      if (name == null) {
         // synchronized
         name = getAttribute2("name");
         if (name == null) {
            throw new RuntimeEventException("error.semantic",
                                    "no name in var");
         }
      }
      return name;
   }
}
```

```
package com.genmagic.invui.vxml.domimpl;
import java.util.Vector;
import java.util.Hashtable;
import org.w3c.dom.Node;
import org.w3c.dom.Attr;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.EventHandlerSet;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLDocumentImpl;
import com.genmagic.invui.vxml.interp.Grammar;
import com.genmagic.invui.vxml.interp.GrammarSet;
import com.genmagic.invui.vxml.interp.NotImplementedException;
import com.genmagic.invui.vxml.dom.Dialog;
import com.genmagic.invui.vxml.dom.Vxml;
import com.genmagic.invui.vxml.dom.Catch;
import com.genmagic.invui.vxml.dom.Link;
import com.genmagic.invui.vxml.dom.Form;
import com.genmagic.invui.vxml.dom.Menu;
import com.genmagic.invui.vxml.dom.Var;
import com.genmagic.invui.vxml.dom.VXMLElement;
import com.genmagic.invui.vxml.dom.Property;
public class TXVxml extends Vxml {
   private Hashtable dialogs = null;
   private Dialog firstDialog = null;
   private Dialog anonymousDialog = null;
   private String application = null;
   private EventHandlerSet set = null;
   private Grammar[] _grammars = null;
   /**
    * For VoiceXML0.9 (1.0 and 0.9 shares only VXML implimentation)
    */
   public Var[] getVarDecls() {
      return VXMLDocumentImpl.getVarDecls(this);
   }
   public VXMLElement[] getDecls() {
      // called only once
      Vector v = new Vector();
      for(Node child = getFirstChild();
         child != null;
         child = child.getNextSibling()) {
         if (child instanceof Var ||
            child instanceof Property) {
            v.addElement(child);
         }
      }
      VXMLElement[] decls = new VXMLElement[v.size()];
      v.copyInto(decls);
      return decls;
   }
/*
   public void setAttribute(String name, String value) {
      System.out.println("setAttribute(" + name + "," + value + ");");
      super.setAttribute(name, value);
   }
   public Attr setAttributeNode(Attr attr) {
      System.out.println("setAttributeNode(" + attr + ")");
```

```
        return super.setAttributeNode(attr);
    }
*/
    /**
     * @deprecated as of 1.0
     */
    public String getProduct() {
        throw new NotImplementedException("product is deprecated as of 1.0");
    }
    public String getApplication() {
        if (dialogs == null) {
            update();
        }
        return application;
    }
    public String getBase() {
        return getAttribute2("base");
    }
    public String getLang() {
        return getAttribute2("lang");
    }
    public String getVersion() {
        return getAttribute2("version");
    }
    public Catch getEventHandler(VXMLInterp ip, Exception ev, int count) throws EventException {
        if (set == null) {
            // synchronized
            set = new EventHandlerSet(this);
        }
        return set.getEventHandler(ip, ev, count);
    }
    public Dialog getDialog(String id) {
        if (dialogs == null) {
            update();
        }
        if (id == null) {
            return firstDialog;
        } else {
            Dialog d = (Dialog)dialogs.get(id);
            if (d == null) {
                return anonymousDialog;
            } else {
                return d;
            }
        }
    }
    synchronized private void update() {
        if (dialogs == null) {
            // synchronized
            dialogs = new Hashtable();
            for (Node child = getFirstChild();
                 child != null;
                 child = child.getNextSibling()) {
                if (child instanceof Dialog) {
                    Dialog d = (Dialog)child;
                    if (firstDialog == null) {
```

```
                firstDialog = d;
              }
              String id_tmp = d.getID();
              // null means invisible?
              if (id_tmp != null) {
                if ("__any".equals(id_tmp) ||
                  // for backward compability with JDE1.0
                  "*".equals(id_tmp)) {
                  anonymousDialog = d;
                } else {
                  dialogs.put(id_tmp, d);
                }
              }
            }
          }
        }
        application = getAttribute2("application");
      }
      public void getDocumentGrammars(VXMLInterp ip, GrammarSet set) throws EventException {
        if (_grammars == null) {
          Vector v = new Vector();
          for (Node child = getFirstChild();
              child != null;
              child = child.getNextSibling()) {
            /*
             * the pre registered grammar will not be used anyway.
             if (child == current) {
             continue;
             } else
             */
            if (child instanceof Link) {
              Link l = (Link)child;
              Grammar[] g = l.getLinkGrammars(ip);
              for (int i=0; i<g.length; i++) {
                v.addElement(g[i]);
              }
            } else if (child instanceof Dialog) {
              Grammar[] g = ((Form)child).getDocumentGrammars(ip);
              for (int i=0; i<g.length; i++) {
                v.addElement(g[i]);
              }
              /*
               Menu menu = (Menu)child;
               if (menu.getScope() == DOCUMENT_GRAMMAR) {
               v.addElement(menu.getGrammar());
               }
               */
            }
          }
          _grammars = new Grammar[v.size()];
          v.copyInto(_grammars);
        }
        set.addGrammars(_grammars);
      }
    /*
      public void setProperty(String name, Object obj) {
```

```
        if (_prop == null) {
            _prop = new Hashtable();
        }
        _prop.put(name, obj);
    }
*/
}
```

Document Page

```
package com.genmagic.invui.vxml.script.expr;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
/** interface for unary function in expressions
 * @see Expression
 */
interface UnaryFunction extends Function {
   Object perform(VXMLInterp ip, Object arg1) throws EventException;
}
```

Google Inc. Exhibit 2010, Page 2 of 3

```
package com.genmagic.invui.vxml.interp;
import java.util.Vector;
import java.util.Hashtable;
import java.util.StringTokenizer;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.text.ParsePosition;
import java.text.NumberFormat;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.prompt.CompositePrompt;
import com.genmagic.invui.vxml.prompt.PromptGenerator;
import com.genmagic.invui.vxml.prompt.PromptGeneratorFactory;
import com.genmagic.invui.vxml.prompt.InvalidFormatException;
import com.genmagic.invui.vxml.dom.Form;
import com.genmagic.invui.vxml.dom.Dialog;
import com.genmagic.invui.vxml.dom.Menu;
import com.genmagic.invui.vxml.dom.Field;
import com.genmagic.invui.vxml.dom.FormItem;
import com.genmagic.invui.vxml.dom.Initial;
import com.genmagic.invui.vxml.dom.Vxml;
import com.genmagic.invui.vxml.dom.VXMLElement;
import com.genmagic.invui.vxml.dom.Expr;
public class Util implements VXMLConstants {
  public static Node[] getPromptElements(Node node) {
    // synchronized
    Vector v = new Vector();
    for (Node child = node.getFirstChild();
        child != null;
        child = child.getNextSibling()) {
      if (child.getNodeType() == Node.TEXT_NODE) {
        v.addElement(child);
      } else {
        String tag = child.getNodeName();
        if ("prompt".equals(tag) ||
            "audio".equals(tag) ||
            "value".equals(tag) ||
            "enumerate".equals(tag)) {
          v.addElement(child);
        }
      }
    }
    Node[] prompts = new Node[v.size()];
    v.copyInto(prompts);
    return prompts;
  }
  static public String[] toArray(String str) {
    StringTokenizer st = new StringTokenizer(str, " ,\t", false);
    Vector v = new Vector();
    while (st.hasMoreTokens()) {
      v.addElement(st.nextToken());
    }
    String[] ret = new String[v.size()];
    v.copyInto(ret);
    return ret;
```

```
    }
    static public String[] copyVariablesFromRecResult(VXMLInterp ip,
                                       FormItem[] items,
                                       RecResult r)
throws EventException {
        Vector v = new Vector();
        for (int i=0; i<items.length; i++) {
            FormItem item = items[i];
            if (item instanceof Field) {
                Field field = (Field)item;
                String slotName = field.getSlot();
                // System.out.print("checking ... " + slotName);
                SlotResult sr = r.getResult(slotName);
                // Object value = s.getValue(slotName);
                if (sr != null) {
                    //  System.out.print(", value = " + value);
                    String varName = field.getName();
                    v.addElement(varName);
                    ip.setVariable(varName, sr.value);
                    ip.createVariable(varName + "$.confidence",
                                new Double(sr.confidence));
                    ip.createVariable(varName + "$.utterance",
                                sr.utterance);
                }
                //System.out.println("");
            }
        }
        String[] varNames = new String[v.size()];
        v.copyInto(varNames);
        return varNames;
    }
    public static GrammarSet getActiveGrammars(VXMLInterp ip, Menu menu) throws EventException {
        GrammarSet set = new GrammarSet();
        // Hashtable h = new Hashtable();

        if (menu.getScope() != DOCUMENT_GRAMMAR) {
            menu.getGrammar(ip, set);
/*
            Grammar[] gram = menu.getGrammar(ip);
            for (int i=0; i<gram.length; i++) {
                h.put(gram[i].getUniqID(), gram[i]);
            }
*/
        }
        // loadDocumentAndApplicationGrammars(h, ip, menu);
        loadDocumentAndApplicationGrammars(set, ip, menu);
        Grammar gram = BuiltInGrammar.getGrammarByType("help");
        set.addGrammar(gram);
        gram = BuiltInGrammar.getGrammarByType("cancel");
        set.addGrammar(gram);
        // set.resolve(ip);
        return set;
    }
    public static GrammarSet getActiveGrammars(VXMLInterp ip, FormItem item) throws EventException {
        GrammarSet set = new GrammarSet();
        boolean modal = false;
```

```
    if (item instanceof Field) {
      Field field = (Field)item;
      modal = field.getModal();
      set.addGrammars(field.getGrammars(ip));
    } else if (item instanceof Initial) {
      Initial init = (Initial)item;
      // Link
      set.addGrammars(init.getGrammars(ip));
    }
    if (modal == false) {
      Form form = (Form)item.getParentNode();
      form.getFormGrammars(ip, set);
      loadDocumentAndApplicationGrammars(set,
                                         ip,
                                         form);
    }
    if (set.size() == 0) {
      throw new EventException("error.semantic",
                               "no grammars active");
    }
    // if (modal == false) { // to get comeback work for now.
      // default built-in
    String str = ip.getCurrentProperties().getProperty("builtin.activegrammars", null);
    Grammar gram = null;
    if (str == null || str.indexOf("help")>=0) {
      gram = BuiltInGrammar.getGrammarByType("help");
      set.addGrammar(gram);
    }
    if (str == null || str.indexOf("cancel")>=0) {
      gram = BuiltInGrammar.getGrammarByType("cancel");
      set.addGrammar(gram);
    }
    //}
    // set.resolve(ip);
    return set;
  }
  // static private void loadDocumentAndApplicationGrammars(Hashtable h,
  static private void loadDocumentAndApplicationGrammars(GrammarSet set,
                                         VXMLInterp ip,
                                         Dialog current) throws EventException {
    Vxml vxml = (Vxml)current.getParentNode();
    vxml.getDocumentGrammars(ip, set);
/*
    Grammar[] grammars = vxml.getDocumentGrammars();
    addGrammars(h, ip, grammars);
*/
    AppContext ac = ip.getAppContext();
    if (ac.isInApplicationRoot() == false) {
      ac.getApplicationGrammars(ip, set);
/*
      grammars = ac.getApplicationGrammars();
      if (grammars != null) {
        addGrammars(h, ip, grammars);
      }
*/
    }
```

```
}
/**
 *
static protected
void addGrammars(Hashtable h, VXMLInterp ip, Grammar[] grammars) throws EventException {
   Grammar gram = null;
   for (int i=0; i<grammars.length; i++) {
      gram = grammars[i];
      if (gram instanceof ExternalGrammar) {
         gram = ip.loadGrammar((ExternalGrammar)gram);
      }
      if (h.containsKey(gram.getUniqID()) == false) {
         // System.out.println("adding.. " + gram);
         h.put(gram.getUniqID(), gram);
      } else {
         // System.out.println("******* Skipping grammar.. " + gram);
      }
   }
}
 */
/**
 * REMIND: this needs to be turned.
 */
public static String resolve(String url, String src) throws java.io.IOException {
   try {
      XMLLoader loader = new com.genmagic.invui.vxml.dialogengine.XMLLoaderImpl(url);
      loader.setPath(src);
      return loader.getPath();
   } catch (Exception ex) {
      ex.printStackTrace();
      return null;
   }
}
public static void main(String a[]) throws Exception {
   System.out.println(resolve(a[0], a[1]));
}
public static String genKey(String text) {
   byte[] bytes = text.getBytes();
   try {
      MessageDigest sha = MessageDigest.getInstance("MD5");
      //return toPrintableString(sha.digest(bytes));
      return toHexString(sha.digest(bytes));
   } catch (NoSuchAlgorithmException ex) {
      try {
         MessageDigest sha = MessageDigest.getInstance("SHA");
         return toHexString(sha.digest(bytes));
      } catch (NoSuchAlgorithmException exx) {
      }
   }
   return toHashString(text);
}
static private String toHashString(String str) {
   int len = str.length();
   long key = len;
   int shift = 0;
   for (int i=0; i<len; i++) {
```

```
      char c = str.charAt(i);
      key += c << shift;
      shift ++;
      if (shift == 48) {
        shift = 0;
      }
    }
    return String.valueOf(Math.abs(key));
  }
  static String toPrintableString(byte[] b) {
    return toPrintableString(new BigInteger(1, b));
  }
  static BigInteger NF = BigInteger.valueOf(94L);
  static String toPrintableString(BigInteger bint) {
    StringBuffer buf = new StringBuffer();
    BigInteger mod = null;
    while (bint.compareTo(NF) >= 0) {
      mod = bint.mod(NF);
      bint = bint.subtract(mod).divide(NF);
      buf.append( (char)(33 + mod.intValue()) );
    }
    buf.append( (char)(33 + bint.intValue()) );
    return buf.toString();
  }
  public static String toPrintableString2(byte[] b) {
    StringBuffer buf = new StringBuffer();
    long val[] = new long[b.length/7 + 1];
    int n = 0;
    for (int pos =0; pos < b.length;) {
      long v = 0;
      for (int i=0; i<7 && pos<b.length; i++, pos++) {
        int ib = b[pos];
        v = v << 8;
        v += ib >=0 ? ib : 127 - ib;
      }
      val[n++] = v;
    }
    for (int i=0; i<val.length; i++) {
      toPrintableString(buf, val[i]);
    }
    return buf.toString();
  }
  static public void toPrintableString(StringBuffer buf, long v) {
    long mod = 0;
    while (v >= 94) {
      mod = v % 94;
      v = (v - mod)/94;
      buf.append( (char)(33 + mod) );
    }
    buf.append( (char)(33 + v) );
  }
  static public String toHexString(byte[] b) {
    StringBuffer buf = new StringBuffer();
    for (int i=0; i<b.length; i++) {
      hexDigit(buf, b[i]);
    }
```

```
    return buf.toString();
}
private static void hexDigit(StringBuffer buf, byte x) {
        char c;

        c = (char) ((x >> 4) & 0xf);
        if (c > 9) {
            c = (char) ((c - 10) + 'a');
        } else {
            c = (char) (c + '0');
        }
        buf.append(c);
        c = (char) (x & 0xf);
        if (c > 9) {
            c = (char)((c - 10) + 'a');
        } else {
            c = (char)(c + '0');
        }
        buf.append(c);
}
public static long parseTime(String str) throws RuntimeEventException {
    ParsePosition pos = new ParsePosition(0);
    int len = str.length();
    long base = 1;
    String nm = str;
    // REMIND:
    // this must first parse it and examine the rest of string.
    // will be fixed later.
    if (len>2) {
        char c = str.charAt(len-2);
        if (c == 'i' && str.endsWith("min")) {
            nm = str.substring(0, len-3);
            base = 60000;
        } else if (c == 'n') {
            if (str.endsWith("mins")) {
                nm = str.substring(0, len-4);
                base = 60000;
            } else if (str.endsWith("second")) {
                nm = str.substring(0, len-6);
                base = 1000;
            }
        } else if ( c == 'd' && str.endsWith("seconds")) {
            nm = str.substring(0, len-7);
            base = 1000;
        } else if ( c == 'e') {
            if (str.endsWith("sec")) {
                nm = str.substring(0, len-3);
                base = 1000;
            } else if (str.endsWith("minutes")) {
                nm = str.substring(0, len-7);
                base = 60000;
            } else if (str.endsWith("msec")) {
                nm = str.substring(0, len-4);
                base = 1;
            }
        } else if ( c == 't' && str.endsWith("minute")) {
```

```
            nm = str.substring(0, len-6);
            base = 6000;
          } else if ( c == 'c') {
            if (str.endsWith("secs")) {
              nm = str.substring(0, len-4);
              base = 1000;
            } else if (str.endsWith("msecs")) {
              nm = str.substring(0, len-5);
              base = 1;
            }
          } else if ( c == 'u' && str.endsWith("hour")) {
            nm = str.substring(0, len-4);
            base = 3600000;
          } else if ( c == 'r' && str.endsWith("hours")) {
            nm = str.substring(0, len-5);
            base = 3600000;
          } else if (c == 'm' && str.endsWith("ms")) {
            nm = str.substring(0, len-2);
            base = 1;
          } else {
            c = str.charAt(len-1);
            if (c == 's') {
              nm = str.substring(0, len-1);
              base = 1000;
            } else if (c == 'm') {
              nm = str.substring(0, len-1);
              base = 1;
            } else {
              nm = str;
              base = 1;
            }
          }
        } else {
          char c = str.charAt(len-1);
          if (c == 's') {
            nm = str.substring(0, len-1);
            base = 1000;
          } else if (c == 'm') {
            nm = str.substring(0, len-1);
            base = 1;
          } else {
            nm = str;
            base = 1;
          }
        }
        Number number = NumberFormat.getInstance().parse(nm.trim(),pos);
        if (pos.getIndex() == 0) {
          throw new RuntimeEventException("error.semantic",
                                  "Invalid Time Format '" + str + "'");
        }
        return number.longValue() * base;
    }
    static public void appendPrompt(Object obj, String klass, CompositePrompt set) throws EventException {
        // ignore klass for the moment.
        if (obj instanceof Promptable) {
          ((Promptable)obj).appendPrompt(klass, set);
```

```
    } else if (klass == null ||
            "string".equals(klass)) {
        set.appendTTS(obj.toString());
    } else {
        appendPrompt(obj.toString(), klass, set);
    }
}
static public void appendPrompt(String content, String klass, CompositePrompt set) throws EventException {
    PromptGenerator gen = PromptGeneratorFactory.getPromptGenerator(klass);
    gen.appendPrompt(set, content);
}
static private Hashtable object_handlers = new Hashtable();
public static ObjectHandler getObjectHandler(String id) throws EventException {
    int i = id.indexOf(":");
    if (i<0) {
        throw new EventException("error.senmatic",
                        "id does not have scheme " + id);
    }
    String klass = id.substring(0, i);
    ObjectHandler handler = (ObjectHandler)object_handlers.get(klass);
    if (handler == null) {
        String klassname = "com.genmagic.invui.vxml.object." + klass + ".Handler";
        try {
            Class cls = Class.forName(klassname);
            Object obj = cls.newInstance();
            if (obj instanceof ObjectHandler == false) {
                throw new EventException("not object handler class :" +
                                klassname);
            } else {
                handler = (ObjectHandler)obj;
                object_handlers.put(klass, handler);
            }
        } catch (ClassNotFoundException ex) {
            throw new EventException("error.semantic",
                            "object handler not found :" +
                            klassname);
        } catch (InstantiationException ex) {
            throw new EventException("error.semantic",
                            "cannot instantiate object handler :"+
                            klassname);
        } catch (IllegalAccessException ex) {
            throw new EventException("error.semantic",
                            "cannot instantiate objet handler :"+
                            klassname);
        }
    }
    return handler;
}
/**
 * Those should be moved to VXMLElementImpl once we move dom classes
 * into interfaces.
 */
static public int getIntegerAttribute(VXMLElement e, String name,
                            int defaultValue) {
    String str = e.getAttribute2(name);
    if (str != null) {
```

```
        try {
          return Integer.parseInt(str);
        } catch (Exception ex) {
          throw new RuntimeEventException("error.semantic",
                                          "invalid integer format in " +
                                          name + " : " + str);
        }
      }
      return defaultValue;
    }
    static public int getTimeAttribute(VXMLElement e, String name,
                                       int defaultValue) {
      String str = e.getAttribute2(name);
      if (str != null) {
        try {
          return (int)Util.parseTime(str);
        } catch (Exception ex) {
          throw new RuntimeEventException("error.semantic",
                                          "invalid time format in " +
                                          name + " : " + str);
        }
      }
      return defaultValue;
    }
    static public boolean getBooleanAttribute(VXMLElement e, String name,
                                              boolean defaultValue) {
      String str = e.getAttribute2(name);
      if (str != null) {
        if ("true".equals(str)) {
          return true;
        } else if ("false".equals(str)) {
          return false;
        }
        throw new RuntimeEventException("error.semantic",
                                        "invalid boolean format in " +
                                        name + " : " + str);
      }
      return defaultValue;
    }
    static public Expr getExprAttribute(VXMLElement e, String name) throws EventException {
      String str = e.getAttribute2(name);
      if (str != null) {
        return new ExprImpl(str);
      }
      return null;;
    }
    /**
     * BASE 64 Encode/Decode
     */
    static public String encodeBASE64(byte[] buf) {
      return (new sun.misc.BASE64Encoder()).encode(buf);
    }
    static public byte[] decodeBASE64(String buf) throws java.io.IOException {
      return (new sun.misc.BASE64Decoder()).decodeBuffer(buf);
    }
}
```

```
package com.genmagic.invui.vxml.dom;
import com.genmagic.invui.vxml.interp.EventException;
abstract public class Value extends VXMLElement {
   public Value() {
      super("value");
   }
   /**
    * the name
    * @deprecated as of Vxml1.0
    */
   abstract public String getName();
   /**
    * the name
    * @since Vxml1.0
    */
   abstract public Expr getExpr() throws EventException;
   /**
    * Get type to which this variable will be converted
    * default is "string"?
    */
   abstract public String get_Class();
   /**
    *
    */
   abstract public String getMode();
   /*
    *
    */
   abstract public String getRecsrc();
}
```

```
package com.genmagic.invui.vxml.tags;
import java.lang.Object;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.dom.Value;
import com.genmagic.invui.vxml.dom.Expr;
import com.genmagic.invui.vxml.interp.TagHandler;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.VXMLException;
import com.genmagic.invui.vxml.interp.Util;
public class ValueTag implements TagHandler {
   public void perform(VXMLInterp ip, Element self) throws VXMLException {
      Value value = (Value)self;
      String klass = value.get_Class();
      Expr expr = value.getExpr();
      if (expr == null) {
         throw new EventException("error.semantic",
                        "no expression in value");
      }
      Object v = ip.evalExpression(expr);
      if (ip.isUndefined(v)) {
         throw new EventException("error.semantic",
                        "variable not defined :" +
                        expr.getExpression());
      }
      Util.appendPrompt(v, klass, ip);
   }
}
```

```
package com.genmagic.invui.vxml.dom;
import com.genmagic.invui.vxml.interp.EventException;
abstract public class Var extends VXMLElement {
   public Var() {
      super("var");
   }
   /**
    * Get expression.
    */
   abstract public Expr getExpr() throws EventException;
   /*
    * Gets the name of variable
    */
   abstract public String getName();
}
```

```
package com.genmagic.invui.vxml.tags;
import org.w3c.dom.Element;
import com.genmagic.invui.vxml.dom.Var;
import com.genmagic.invui.vxml.dom.Expr;
//import com.genmagic.invui.vxml.interp.Expr;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLException;
import com.genmagic.invui.vxml.interp.TagHandler;
public class VarTag implements TagHandler {
   public void perform(VXMLInterp ip, Element self) throws VXMLException {
      Var var = (Var)self;
      //Expr expr = var.getExpr();
      Expr expr = var.getExpr();
      if (expr == null) {
         ip.createVariable(var.getName());
      } else {
         ip.createVariable(var.getName(),
                  ip.evalExpression(var.getExpr()));
         // ip.evalExpression(var.getExpr()));
      }
   }
}
```

```
package com.genmagic.invui.vxml.dialogengine;
import com.genmagic.invui.vxml.idl.invui.*;
import org.omg.CORBA.*;
public class VUIBrokerImpl extends VUIBrokerPOA {
   ORB orb;
   public VUIBrokerImpl(ORB o) {
      orb = o;
   }
   public Call placeCall(String number,
                         String ani,
                         byte[] uui,
                         int timeout,
                         Property[] props) throws NoMoreChannel, UserBusy, NetworkBusy, CallRejected, InvalidNumber,
NetworkError, TelephonyError, Timeout
   {
      VUIClient client = getVUIClient();
      MTSServer mts = getMTSServer();
      return mts.placeCall(client, number, ani, uui, timeout);
   }
   public void updateAvailableLines(MTSServer m, short numberOfAvailableLines) {
      String id = orb.object_to_string(m);
   }
   public void updateHandlingCalls(VUIClient vc, short numberOfHandlingCalls) {
      String id = orb.object_to_string(vc);
   }
   public void unregisterVUIClient(VUIClient vc) {
      String id = orb.object_to_string(vc);
   }
   public void unregisterMTS(MTSServer m) {
      String id = orb.object_to_string(m);
   }
   public VUIClient findVUIClient(Call c) {
      return null;
   }
   public VUIClient getVUIClient() {
      return null;
   }
   public MTSServer getMTSServer() {
      String group = "HOTEI";
      return MTSServerHelper.bind(orb, "/MTS_poa",
                           ("MTS." + group).getBytes());
   }
};
```

```
package com.genmagic.invui.vxml.dialogengine;
import java.util.Hashtable;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import com.genmagic.invui.util.Log;
import com.genmagic.invui.util.LogFactory;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.idl.invui.*;
public class VUIClientImpl extends VUIClientPOA {
   private DialogEngine engine;
   private String mts_group = null;
   private Hashtable outgoings = new Hashtable();
   private ORB orb = null;
   static final Log log = LogFactory.getLogFor("jde");
   VUIClientImpl(DialogEngine e, ORB o, String mg) {
      engine = e;
      orb = o;
      mts_group = mg;
   }
   public Call placeCall(String number, String sub, byte[] uui, int timeout) throws Exception {
      log.debug("connecting MTS." + mts_group);
      MTSServer mts =
         MTSServerHelper.bind(orb, "/MTS_" + mts_group + "_poa",
                  ("MTS." + mts_group).getBytes());
      if (mts == null) {
         throw new Exception("Cannot Find MTS : " + mts_group);
      }
      VUIClient client =
         VUIClientHelper.narrow(_default_POA().servant_to_reference(this));
      log.info("making outgoing call number = " + number + " ani=" + sub);
      Call call = mts.placeCall(client, number, sub, uui, timeout);
      waitForCallConnection(call);
      return call;
   }
   void waitForCallConnection(Call call) throws Timeout, InvalidCall, SessionEnded {
      String id = orb.object_to_string(call.session);
      OutgoingCallRequest ocr = new OutgoingCallRequest();
      outgoings.put(id, ocr);
      log.diag("[" + call.id + "] waiting for connection ");
      boolean connected = ocr.waitForConnection(40000);
      // current ++;
      outgoings.remove(id);
      if (connected == false) {
         log.warning("[" + call.id + "] cancelling connection");
         try {
            if (call.session.cancelCall(call)==false) {
               call.session.terminate();
            }
         } catch (Exception ex) {
            log.diag("exception when terminating call :"+
                  ex.getMessage());
         }
         throw new Timeout("not connected");
      }
      log.info("[" + call.id + "] got connected ");
   }
```

```
  void waitForCallTermination(Call call) throws Timeout {
  }
  synchronized public void ping() {
    try {
      wait(10000); // every 10sec
    } catch (Exception ex) { }
  }
  public boolean handleCall(Call c, String dnis, String ani, byte[] UUI) {
    return engine.handleCall(c, dnis, ani, UUI);
  }
  public void callDropped(Call c) {
    VXMLInterpCORBAImpl s = engine.removeSession(c);
    if (s != null) {
      s.callDropped(c);
    }
  }
  public void sessionEnded(VUISession s) {
    log.info("sessionEnded");
  }
  public short incomingCapacity() {
    log.info("incomingCapacity");
    return 10;
    // return max;
  }
  public void callConnected(Call c) {
    VXMLInterpCORBAImpl s = engine.getSession(c);
    if (s != null) {
      s.callConnected(c);
      return;
    }
    log.info("[" + c.id + "] call connected ");
    String id = orb.object_to_string(c.session);
    OutgoingCallRequest ocr =
      (OutgoingCallRequest)outgoings.get(id);
    if (ocr == null) {
      synchronized(this) {
        try {
          wait(100);
        } catch (Exception ex) { }
      }
/*
      Thread.yield();
      Thread.yield();
      Thread.yield();
*/
      ocr = (OutgoingCallRequest)outgoings.get(id);
    }
    if (ocr != null && ocr.callConnected()) {
      return;
    }
    // timeout
    try {
      c.session.terminate();
    } catch (Exception ex) {
    }
/*
```

```
        CallSession s = getCallSession(c);
        if (s != null) {
          s.info("callConnected");
        }
*/
    }
    public void callCancelled(Call c) {
        log.info("[" + c.id + "] call cancelled");
        String id = orb.object_to_string(c.session);
        OutgoingCallRequest ocr =
          (OutgoingCallRequest)outgoings.get(id);
        if (ocr != null && ocr.callCancelled()) {
          return;
        }
/*
        CallSession s = getCallSession(c);
        if (s != null) {
          s.info("callConnected");
        }
*/
    }
    public void callsDisconnected(Call c1, Call c2) {
        VXMLInterpCORBAImpl s = engine.getSession(c1);
        if (s != null) {
          s.callsDisconnected(c1, c2);
        }
    }
    public void dtmfPressed(Call c, char dtmf) {
        VXMLInterpCORBAImpl s = engine.getSession(c);
        if (s != null) {
          s.dtmfPressed(c, dtmf);
        }
    }
};
```

```
package com.genmagic.invui.vxml.dom;
import org.w3c.dom.*;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.EventException;
import com.genmagic.invui.vxml.interp.GrammarSet;
abstract public class Vxml extends VXMLElement {
  public Vxml() {
    super("vxml");
  }
  /**
   * Application Root
   */
  abstract public String getApplication();
  /**
   * Get base
   * @since VXML1.0
   */
  abstract public String getBase();
  /**
   * Get Language information
   * @since VXML1.0
   */
  abstract public String getLang();
  /**
   * Get VXML version
   * @since VXML1.0
   */
  abstract public String getVersion();
  /**
   * Var elements in this form
   * @deprecated as of VXML1.0
   */
  abstract public Var[] getVarDecls();
  /**
   * Get declarations
   * @since VXML1.0
   */
  abstract public VXMLElement[] getDecls();
  /**
   * Get Dialog
   */
  abstract public Dialog getDialog(String name);
  /*
   * Gets event handler (catch)
   */
  abstract public Catch getEventHandler(VXMLInterp ip, Exception ev, int count) throws EventException;
  /**
   * Get active grammar definitions on this form item.
   */
  abstract public void getDocumentGrammars(VXMLInterp ip, GrammarSet set) throws EventException;
}
```

```
package com.genmagic.invui.vxml.interp;
public interface VXMLConstants {
  /*
   * TRANSFER
   */
  static final public int TR_CONNECTED = 0;
  static final public int TR_FAR_END_DISCONNECTED = 1;
  static final public int TR_NEAR_END_DISCONNECTED = 2;
  static final public int TR_NETWORK_DISCONNECTED = 3;
  static final public int TR_BUSY = 4;
  static final public int TR_NETWORK_BUSY = 5;
  static final public int TR_NOANSWER = 6;
  static final public int TR_ERROR = 7;
  /**
   * Bargein Value
   */
  static final public int BARGEIN_NONE = -1;
  static final public int BARGEIN_OFF = 0;
  static final public int BARGEIN_ON = 1;
  /**
   * Variable Scope
   */
  static final public int SESSION = 0;
  static final public int APPLICATION = 1;
  static final public int DOCUMENT = 2;
  static final public int DIALOG = 3;
  static final public int LOCAL = 3; // for backword compatibility
  /**
   * Grammar Scope
   */
  static final public int LOCAL_GRAMMAR    = 0;
  static final public int DOCUMENT_GRAMMAR = 1;
  /**
   * HTTP Method
   */
  static final public int POST = 0;
  static final public int GET = 1;
  static final String METHOD_STRING[] = {
    "GET",
    "POST"
  };
  /**
   * Cache
   */
  final static public int SAFE = 0;
  final static public int FAST = 1;
  static final String CACHING_STRING[] = {
    "SAFE",
    "FAST"
  };
  static public final int PREFETCH = 1;
  static public final int STREAM = 2;
  static final String FETCHHINT_STRING[] = {
    "SAFE",
    "PREFETCH",
    "STREAM"
```

```
};
/**
 * Default Parameters
 */
static final public int DEFAULT_MAX_DIGIT = 0;
public static final int DEFAULT_TOOMUCH_TIMEOUT = 10000;
public static final int DEFAULT_REC_TIMEOUT = 3000;
public static final int DEFAULT_ENDSPEECH_TIMEOUT = 750;
public static final int DEFAULT_RECORD_MAX_LENGTH = 3000;
public static final int DEFAULT_RECORD_TIMEOUT = 60000;
public static final int DEFAULT_FETCH_TIMEOUT = 10000;
/**
 * CONDITION of NEXT DOCUMENT
 */
public static final int IN_DOCUMENT = 0;
public static final int EXTERNAL = 1;
public static final int IN_APPLICATION_ROOT = 2;
public static final int ANOTHER_DOC_IN_APPLICATION = 3;
/**
 * FOR TELEPHONY
 */
public final static int CALL_CONNECTED = 0;
public final static int CALL_BUSY     = 1;
public final static int CALL_TIMEOUT   = 2;
public final static int CALL_NO_ANSWER = 3;
public final static int CALL_ERROR     = 4;

public final static String CALL_STATUS[] = {
   "CONNECTED",
   "BUSY",
   "TIMEOUT",
   "NO_ANSWER",
   "ERROR",
 };
}
```

```
package com.genmagic.invui.vxml.domimpl;
import java.util.Hashtable;
import com.genmagic.invui.vxml.tags.*;
import com.genmagic.invui.vxml.dom.VXMLElement;
import com.genmagic.invui.vxml.interp.TagHandler;
public class VXMLDefs {
  private static void add(Hashtable t, String name, VXMLElement proto, TagHandler h) {
    proto.setTagHandler(h);
    t.put(name, proto);
  }
  private static void add(Hashtable t, String name, VXMLElement proto) {
    t.put(name, proto);
  }
  static public Hashtable getPrototypes() {
    Hashtable h= new Hashtable();
    TagHandler catchTag = new CatchTag();
    add(h, "assign", new TXAssign(), new AssignTag());
    add(h, "audio", new TXAudio(), new AudioTag());
    add(h, "block", new TXBlock(), new BlockTag());
    add(h, "break", new TXBreak(), new BreakTag());
    add(h, "cancel", new TXCatch("cancel", "cancel"), catchTag);
    add(h, "catch", new TXCatch(), catchTag);
    add(h, "clear", new TXClear(), new ClearTag());
    add(h, "choice", new TXChoice());
    add(h, "enumerate", new TXEnumerate(), new EnumerateTag());
    add(h, "error",  new TXCatch("error", "error"), catchTag);
    add(h, "exit", new TXExit(), new ExitTag());
    add(h, "field", new TXField(), new FieldTag());
    add(h, "filled", new TXFilled(), new FilledTag());
    add(h, "form", new TXForm(), new FormTag());
    add(h, "goto", new TXGoto(), new GotoTag());
    add(h, "grammar", new TXGrammar());
    add(h, "if", new TXIf(), new IfTag());
    add(h, "initial", new TXInitial(), new InitialTag());
    add(h, "help", new TXCatch("help", "help"), catchTag);
    add(h, "link", new TXLink());
    add(h, "menu", new TXMenu(), new MenuTag());
    add(h, "noinput", new TXCatch("noinput", "noinput"), catchTag);
    add(h, "nomatch", new TXCatch("nomatch", "nomatch"), catchTag);
    add(h, "object", new TXObject(), new ObjectTag());
    add(h, "param", new TXParam());
    add(h, "prompt", new TXPrompt(), new PromptTag());
    add(h, "property", new TXProperty(), new PropertyTag());
    add(h, "record", new TXRecord(), new RecordTag());
    add(h, "reprompt", new TXReprompt(), new RepromptTag());
    add(h, "sayas", new TXSayas(), new SayasTag());
    add(h, "submit", new TXSubmit(), new SubmitTag());
    add(h, "throw", new TXThrow(), new ThrowTag());
    add(h, "transfer", new TXTransfer(), new TransferTag());
    add(h, "value", new TXValue(), new ValueTag());
    add(h, "var", new TXVar(), new VarTag());
    add(h, "vxml", new TXVxml(), new VxmlTag());
    return h;
  }
}
```

```
package com.genmagic.invui.vxml.dom;
//import com.ibm.xml.parser.TXDocument;
import org.apache.xerces.dom.DocumentImpl;
abstract public class VXMLDocument extends DocumentImpl {
   abstract public String getSourceURI();
}
```

Document Page

```
package com.genmagic.invui.vxml.interp;
import java.lang.Object;
import java.util.Hashtable;
import java.util.Vector;
import org.w3c.dom.*;
import com.genmagic.invui.vxml.dom.VXMLDocument;
import com.genmagic.invui.vxml.dom.VXMLElement;
import com.genmagic.invui.vxml.dom.Vxml;
import com.genmagic.invui.vxml.dom.Var;
import com.genmagic.invui.vxml.tags.PrintTag;
public class VXMLDocumentImpl extends VXMLDocument implements VXMLConstants {
   static final protected Hashtable _prototype09 =
         com.genmagic.invui.vxml.vxml09.domimpl.VXMLDefs.getPrototypes();
   static protected Hashtable _prototype10 =
      com.genmagic.invui.vxml.domimpl.VXMLDefs.getPrototypes();
   static private TagHandler _vxml09TagHandler = null;
   static private boolean _enable09 = true;
   private boolean checkVersion = false;
   private String url;
   protected Hashtable _prototype = _prototype10;
   static {
      VXMLInterpImpl.addTagHandler("print", new PrintTag());
      VXMLElement elem = (VXMLElement)_prototype09.get("vxml");
      _vxml09TagHandler = elem.getTagHandler();
      _enable09 = Boolean.getBoolean("jde.voicexml09");
   }
   public VXMLDocumentImpl() {
   }
   public VXMLDocumentImpl(String u) {
      url = u;
   }
   public void setSourceURI(String u) {
      url = u;
   }
   public String getSourceURI() {
      return url;
   }
   static public void enableVoiceXML09(boolean b) {
      _enable09 = b;
   }
   public Element createElement(String name) throws DOMException {
         if (errorChecking && !isXMLName(name)) {
        super.createElement(name);
      }
      if (checkVersion) {
         checkVersion = false;
         Vxml vxml = (Vxml)getDocumentElement();
         String version = vxml.getVersion();
         if (_enable09 &&
            (version == null || "0.9".equals(version))) {
            _prototype = _prototype09;
            vxml.setTagHandler(_vxml09TagHandler);
         } else if ("1.0".equals(version)) {
            // do nothing
         } else {
            throw new RuntimeException("VoiceXML version is incorrect." +
```

```
                        version);
      }
    }
    Object obj = _prototype.get(name);
    if (obj == null) {
      return super.createElement(name);
    }
    // not Node.cloneNode(false);
    // using clone requires all component does not have
    // sub object in it, which will be shared after cloning.
    VXMLElement el = (VXMLElement)((VXMLElement)obj).clone();
    el.setFactory(this);
    if (el instanceof Vxml) {
      checkVersion = true;
    }
    return el;
  }
  /**
   * REMIND: move to instance.
   */
  static int varCount = 0;
  public synchronized static String genVarName(String pref) {
    return pref + String.valueOf(varCount ++);
  }
  /*
   *
   */
  public static Var[] getVarDecls(Node node) {
    Vector v = new Vector();
    for(Node child = node.getFirstChild();
      child != null;
      child = child.getNextSibling()) {
      if (child instanceof Var) {
        v.addElement(child);
      }
    }
    Var[] vars = new Var[v.size()];
    v.copyInto(vars);
    return vars;
  }
  public static String[] getNames(String name) {
    if (name == null) {
      return new String[0];
    }
    return Util.toArray(name);
  }
  /*
  static public void main(String a[]) {
    Object obj = new TXField();
    long time;
    time= System.currentTimeMillis();
    VXMLElement vobj = (VXMLElement)obj;
    for (int i=0; i<100000; i++) {
      obj = (VXMLElement)vobj.clone();
    }
    System.out.println(System.currentTimeMillis() - time);
```

```
        time= System.currentTimeMillis();
        for (int i=0; i<100000; i++) {
            obj = new TXField();
        }
        System.out.println(System.currentTimeMillis() - time);
    }
    */
}
```

Document Page 2

```
package com.genmagic.invui.vxml.dom;
import com.genmagic.invui.vxml.interp.TagHandler;
import com.genmagic.invui.vxml.interp.VXMLInterp;
import com.genmagic.invui.vxml.interp.VXMLException;
import com.genmagic.invui.vxml.interp.Util;
import org.w3c.dom.*;
//import com.ibm.xml.parser.TXElement;
import org.apache.xerces.dom.ElementImpl;
import org.apache.xerces.dom.DocumentImpl;
public class VXMLElement extends ElementImpl {
   static protected final String NULL = "null";
   static public final Expr NULL_EXPR = new Expr() {
      public String getExpression() {return "";};
      public java.lang.Object getExpressionObject() {return "";};
      public void setExpressionObject(java.lang.Object obj) {};
   };
   public VXMLElement(String tagName) {
      super(null, tagName);
   }
   public void setFactory(Document doc) {
      this.ownerDocument = (DocumentImpl)doc;
   }
   public java.lang.Object clone() {
      try {
         return super.clone();
      } catch (CloneNotSupportedException ex) {
         ex.printStackTrace();
         return null;
      }
   }
   /****
    * for now
    ****/
   public String getAttribute2(String name) {
      Node attr = getAttributes().getNamedItem(name);
      if (attr == null) {
         return null;
      } else {
         return attr.getNodeValue();
      }
   }
   public void perform(VXMLInterp ip) throws VXMLException {
      if (handler == null) {
         // synchronized
         String name = getTagName();
         handler = (TagHandler)ip.getTagHandler(name);
      }
      if (handler != null) {
         handler.perform(ip, this);
      } else {
         System.out.println("handler not found . " + getNodeName());
      }
   }
   public Form getParentForm() {
      Node node = getParentNode();
      while (node != null && node instanceof Form == false) {
```

```
        node = node.getParentNode();
      }
      return (Form)node;
    }
    public Dialog getParentDialog() {
      Node node = getParentNode();
      while (node != null && node instanceof Dialog == false) {
        node = node.getParentNode();
      }
      return (Dialog)node;
    }
/*
    public TagHandler getTagHandler() {
      return handler;
    }
*/
    public void setTagHandler(TagHandler h) {
      handler = h;
    }
    public TagHandler getTagHandler() {
      return handler;
    }
    private TagHandler handler;
}
```

```
package com.genmagic.invui.vxml.interp;
public class VXMLException extends Exception {
   Throwable _nested = null;
   protected VXMLException() {
   }
   protected VXMLException(String msg) {
      super(msg);
   }
   protected VXMLException(String msg, Throwable t) {
      super(msg);
      _nested = t;
   }
   public Throwable getNestedException() {
      return _nested;
   }
}
```

Document not found

```
package com.genmagic.invui.vxml.interp;
import org.w3c.dom.*;
import com.genmagic.invui.vxml.prompt.CompositePrompt;
import com.genmagic.invui.vxml.dom.Form;
import com.genmagic.invui.vxml.dom.Dialog;
import com.genmagic.invui.vxml.dom.Expr;
/**
 * The interpretr is responsible for executing VoiceXML and
 * maintaining runtime information.
 * - composite prompts appended so far, and related information such as
 *   timeout, bargein, etc.
 * - prompt controls (count/bargein/timeout, etc)
 * - access to speech session object.
 * - application state.
 */
public interface VXMLInterp extends CompositePrompt, Runnable {
    /**
     * Get Speech Control Object
     */
    abstract public SpeechControl getSpeechControl();
    /*=========================================
     * Context/Application Control
     *=========================================*/
    abstract public AppContext getAppContext();
    /**
     * Returns the dialog object which is currently being executed.
     * @return the dialog object which is currently being executed.
     */
    abstract public Dialog getCurrentDialog();
    /*=========================================
     * Variable
     *=========================================*/
    /**
     * Checks if the variable of given name exists in given scope.
     * @param scope the scope of variable to check
     * @param name the name of variable to check
     * @return true if the variable exists in the scope.
     */
    abstract public boolean hasDefinedVariable(int scope, String name);
    /**
     * Gets the value of the variable.
     * @param name the name of variable
     * @return a value of the variable
     * @exception EventException of error.semantic if no variable exists,
     *          or undefined.
     */
    abstract public Object getVariable(String name) throws EventException;
    /**
     * Gets the value of the variable in the given scope.
     * @param scope the scope of variable. This can be, SESSION, APPLICATION,
     *           DOCUMENT or LOCAL in VXMLConstants interface.
     * @param name the name of variable
     * @return a value of the variable
     * @exception EventException with "error.semantic" if variable does not
     *          exist, or undefined.
     * @see VXMLConstants
```

```
abstract public Object getVariable(int scope, String name) throws EventException;
 */
/**
 * Assigns a value to a variable.
 * @param name the name of variable to assign to.
 * @param value the value to be assigned to the variable
 * @exception EventException with "error.semantic" if variable does not
 *          exist.
 */
abstract public void setVariable(String name, Object value) throws EventException;
/**
 * Assigns the value to the variable in the specified scope.
 * @param name the name of variable to assign to.
 * @param value the value to be assigned to the variable
 * @exception EventException with "error.semantic" if variable does not
 *          exist.
abstract public void setVariable(int scope, String name, Object value) throws EventException;
 */
/**
 * Creates a new variable with the given value in the current scope.
 * @param name the name of variable to create.
 * @param value the value to be assigned to the new variable
 */
abstract public void createVariable(String name, Object value) throws EventException ;
/**
 * Creates a new undefined variable in the current scope.
 * @param name the name of variable to create.
 */
abstract public void createVariable(String name) throws EventException;
/**
 * Creates new variables with the same given value in the current scope.
 * @param names the array of variable names to create.
 * @param value the value to be assigned to the new variables
 */
abstract public void createVariables(String names[], Object value) throws EventException;
/**
 * Creates new variables with the same given value in the given scope.
 * @param scope the scope of variables to create.
 * @param names the array of variable names to create.
 * @param value the value to be assigned to the new variables
abstract public void createVariable(int scope, String name, Object value) ;
 */
/**
 * Clear the content of variable. After this, the variable is marked
 * as undefined, and all subsequence access to this variable causes
 * error.semantic event to be thrown.
 * @param name the variable name to clear
 * @exception name EventException with "error.semantic" if the variable
 *          does not exist.
abstract public void clearVariable(String name) throws EventException;
 */
/**
 * Clear the content of variable in specific scope.
 * After this, the variable is marked as undefined, and all subsequence
 * access to this variable causes error.semantic event to be thrown.
 *
```

```
 * @param scope the variable name to clear
 * @param name the variable name to clear
 * @exception name EventException with "error.semantic" if the variable
 *          does not exist.
 */
abstract public void clearVariable(int scope, String name) throws EventException;
/**
 * Clear the content of variables in specific scope
 * After this, variables are marked as undefined, and all subsequence
 * access to those variables causes error.semantic event to be thrown.
 *
 * @param scope the variable name to clear
 * @param names the variable names to clear
 * @exception name EventException with "error.semantic" if one of variables
 *          does not exist.
 */
abstract public void clearVariables(int scope, String names[]) throws EventException;
/**
 * Remove the variables from the current scope
 *
 * @param names the variable names to remove
 * @exception name EventException with "error.semantic" if one of variables
 *          does not exist.
 */
abstract public void removeVariable(String name) throws EventException;
abstract public boolean isNotFound(Object obj);
abstract public boolean isUndefined(Object obj);
/*=========================================
 * Expression
 *=========================================*/
/**
 * Evaluates the expression and returns the result.
 * @param expr the expression object to evaluate
 * @return the result object
 * @return EventException with "error.syntax" if syntax error in
 *                  expression, or "error.semantic" if variable
 *                  used in the expression does not exists, or
 *                  is undefiened.
 */
abstract public Object evalExpression(Expr expr) throws EventException;
/**
 * Evaluates the expression and returns the boolean result.
 * @param expr the expression object to evaluate
 * @return the true if the result is boolean and true, false otherwise.
 * @return EventException with "error.syntax" if syntax error in
 *                  expression, or "error.semantic" if variable
 *                  used in the expression does not exists, or
 *                  is undefiened.
 */
abstract public boolean evalBooleanExpression(Expr expr) throws EventException;
/*=========================================
 * Flow Control
 *=========================================*/
/**
 * Directs the current execution to the next dialog (form/menu).
 * It may load the next document, or simply jump to the other dialog
```

```
 * within the same document. When it fail to find next dialog,
 * it throws the EventException with error.badfetch and remains in the
 * same document.
 * @param next the url of next dialog.
 * @exception EventException with "error.badnext" or "error.badfetch"
 *          if it could not goto the specified url.
 */
abstract public void gotoNext(String next, int caching, int timeout) throws VXMLException;
/**
 * Submits the variables listed in namelist to the server specified in
 * next field and fetch the resulting document. Then, it directs the
 * current execution to the next dialog (form/menu) in that document.
 * It may load the next document, or simply jump to the other dialog
 * within the same document. In latter case, variables would not be
 * sent to the server. When it fail to find next dialog,
 * it throws the EventException with error.badfetch and remains in the
 * same document.
 *
 * @param next the url of next dialog.
 * @param submit an array of variable names to be used when loading
 *           a next document.
 * @param method GET or POST for now.
 * @param enctype the MIME encoding type of submitted document.
 * @param caching SAFE to ensure that it get most recent document, or FAST
 *           to use the cached copy if exists and not expired.
 * @param timeout the interval to wait before it gives up.
 * @see VXMLConstants
 * @exception EventException with "error.badfetch" if it could not goto
 *          the specified url.
 */
abstract public void submit(String next, String namelist[], int method, String enctype, int caching, int timeout)
throws VXMLException;
abstract public void unhandledEvent(EventException ev);
abstract public void unhandledEvent(RuntimeEventException ev);
/*=========================================
 * Scoping
 *=========================================*/
abstract public void beginApplicationScope();
abstract public void beginAnonymousScope();
abstract public void endAnonymousScope();
/**
 * Get the slot which is used as an initial slot value.
 * This will be set when form/document/application level
 * grammar is activated.
 */
abstract public RecResult getInitialDialogRecResult();
/*=========================================
 * Evaluating
 *=========================================*/
/**
 * Evaluates the given node. It looks up the corresponding TagHandler
 * for given node and call the tag handler.
 * @param a node to evaluate
 * @exception EventException
 * @see TagHandler
 */
```

```
abstract public void eval(Node node) throws VXMLException;
/**
 * Evaluates given list of nodes. It iteratively calls the
 * <code>eval(Node)</code> to evaluate all nodes.
 * @param nodes an array of nodes to evaluate
 * @exception EventException
 */
abstract public void eval(Node[] nodes) throws VXMLException;
/**
 * Evaluates child nodes of given node. It iteratively calls the
 * <code>eval(Node)</code> for each child.
 * @param node a node whose children will be evaluated.
 * @exception
 */
abstract public void evalChildren(Node node) throws VXMLException;
/*=========================================
 * Resource control
 *=========================================*/
/**
 * Loads the audio file from given src if necessary, and returns the
 * platform specific filename of that resoruce.
 *
 * @param src the src of audio file. This needs to start with "builtin://"
 * for the time being.
 * @return a platform specific filename of audio file.
 * @exception EventException if file not found.
 */
abstract public String loadAudio(String src) throws EventException;
/**
 * Loads the audio file from given src if necessary, and returns the
 * platform specific filename of that resoruce.
 *
 * @param src the src of audio file. This needs to start with "builtin://"
 * for the time being.
 * @param caching safe or fast
 * @param fetchTimeout -1 if you don not want to timeout.
 * @return a platform specific filename of audio file.
 * @exception EventException if file not found.
 */
abstract public String loadAudio(String src,
                                 int caching,
                                 int fetchTimeout) throws EventException;
/**
 * Loads the grammar file
 *
 * @param src the URL to the external grammar
 * @return the content of grammar.
 * @exception EventException if file not found.
 */
abstract public String loadGrammar(String src,
                                   int caching,
                                   int fetchTimeout) throws EventException;
/**
 * @return true if graceful
 */
abstract public boolean runDialog(String first) throws java.io.IOException, VXMLException;
```

```
    /**
     * Tag Management
     */
    abstract public TagHandler getTagHandler(String name);
    /*
     * Debugging
     */
    abstract public void assert(String msg);
    /*
     * Logging
     */
    abstract public void error(String msg);
    abstract public void warning(String msg);
    abstract public void info(String msg);
    abstract public void diag(String msg);
    abstract public void debug(String msg);
    /**
     * Call Info
     */
    public void setCallInfo(String ANI, String DNIS, byte[] UUI) throws VXMLException ;
    /**
     * Property
     */
    public void setProperty(String name, Object obj)  throws VXMLException;

    public Properties getCurrentProperties();
    /**
     * Property Scope
     */
    public void beginPropertyScope() throws VXMLException;
    /**
     *
     */
    public void endPropertyScope() throws VXMLException;
}
```

```
package com.genmagic.invui.vxml.dialogengine;
import java.util.Vector;
import java.util.Enumeration;
import org.omg.CORBA.*;
import com.genmagic.invui.vxml.interp.*;
import com.genmagic.invui.vxml.prompt.*;
import com.genmagic.invui.vxml.idl.invui.*;
import com.genmagic.invui.vxml.idl.invui.SlotPackage.*;
import com.genmagic.invui.vxml.idl.invui.PromptPackage.*;
/**
 * VUIBroker CORBA Implementation
 */
public class VXMLInterpCORBAImpl extends VXMLInterpImpl implements SpeechControl, MTSConstants {
   /* for transfer */
   // private static final int CALLING = -1;
   private boolean reprompt = false;
   protected VUISession session;
   private Call call;
   private boolean dropped = false;
   private String logPrefix = null;
   private DialogEngine de;
   private TransferHandler trans;
   private boolean _enableLog = false;
   private String _grammarlist = null;
   // private int transferState = TR_FAR_END_DISCONNECTED;
   // private Call trans_call;
   public VXMLInterpCORBAImpl(AppContext app, DialogEngine de, Call c) {
      super(app);
      session = c.session;
      call = c;
      this.de = de;
      logPrefix = "[" + call.id + "] ";
   }
   void check() throws EventException {
      if (dropped) {
         throw new EventException("telephone.disconnect.hangup",
                                  "call dropped");
      }
   }
   Call getCall() {
      return call;
   }
   DialogEngine getDialogEngine() {
      return de;
   }
   boolean isDropped() {
      return dropped;
   }
   synchronized void callDropped(Call droppedCall) {
      if (call.id == droppedCall.id) {
         info("main call dropped " + call.id);
         dropped = true;
         notifyAll();
      } else if (trans != null) {
         trans.callDropped(droppedCall);
         trans = null;
```

```
        }
/*
        } else if (trans_call.id == droppedCall.id) {
          info("transfer call dropped " + trans_call.id);
          transferState = TR_FAR_END_DISCONNECTED;
          trans_call = null;
          notifyAll();
        }
*/
      }
      void callConnected(Call c) {
        synchronized(this) {
          if (trans != null && trans.callConnected(c)) {
            return;
          }
/*
          if (transferState == CALLING &&
            trans_call.id == c.id) {
            transferState = TR_CONNECTED;
            notifyAll();
            return;
          }
*/
        }
        try {
          c.session.terminate();
        } catch (Exception ex) {}
      }
      void callsDisconnected(Call c1, Call c2) {
        info("call disconnected " + c1.id + "," + c2.id);
      }
      void dtmfPressed(Call from, char dtmf) {
        info("dtmfPressed " + dtmf);
        TransferHandler th = trans;
        if (th != null) {
          // do I need to check from?
          th.dtmf(dtmf);
        }
      }
      public SpeechControl getSpeechControl () {
        return this;
      }
      public void setReprompt(boolean b) {
        reprompt = b;
      }
      public boolean getReprompt() {
        return reprompt;
      }
      /**
       * SpeechControl
       */
      public void playPrompt() throws VXMLException {
        try {
          check();
          Prompt[] p = getPrompts();
          clearPrompts();
```

```
        diag("appending:", p);
        session.appendPrompts(p);
        diag("playing prompts");
        session.playPrompts();
        debug("prompts done");
      } catch(SessionEnded ex) {
        throw new EventException("telephone.disconnect.hangup", ex.getMessage());
      } catch(MediaError ex) {
        error("error occured in playPrompt: " + ex.getMessage());
        if (ex.error_code == 3) {
          throw new EventException("error.semantic.noprompt", ex.getMessage());
        } else {
          ex.printStackTrace();
          throw new EventException("error.mts.media", ex.getMessage());
        }
      } catch(UserException ex) {
        error("error occured in playPrompt: " + ex.getMessage());
        ex.printStackTrace();
        throw new EventException("error.mts.corba.user", ex.getMessage());
      } catch(SystemException ex) {
        error("system error occured in playPrompt: " + ex.getMessage());
        ex.printStackTrace();
        throw new EventException("error.mts.corba.system", ex.getMessage());
      }
    }
    /**
     * Plays prompt and start recognition. It returns the result of the
     * recognition. If there is no speech, or no match with grammar,
     * it throws noinput, nomatch event respectively.
     */
    public RecResult playAndRecognition(Grammar grammar, int noinput_timeout,
                                        int toomuch_timeout,
                                        int endspeech_timeout,
                                        int maxdigit) throws VXMLException {
      try {
        check();
        Prompt[] p = composite.getNoBargeinPrompts();
        if (p.length >0) {
          if (_log.isDiagEnabled()) {
            diag("appending with no bargein:", p);
          }
          session.appendPrompts(p);
          composite.clearNoBargeinPrompts();
          diag("playing prompts");
          session.playPrompts();
          debug("playing prompts done");
        }
        p = getPrompts();
        if (p.length != 0) {
          if (_log.isDiagEnabled()) {
            diag("appending:", p);
          }
          session.appendPrompts(p);
        }
        clearPrompts();
/*
```

```
        if (composite.getBargein() == false) {
          diag("playing prompts in rec: bargein=" + composite.getBargein());
          session.playPrompts();
          debug("playing done");
          composite.setBargein(true);
        }
*/
        if (_grammarlist != null) {
          _grammarlist = _grammarlist.trim();
          if (_grammarlist.length() == 0 ||
            "none".equals(_grammarlist) ||
            "empty".equals(_grammarlist)) {
            grammar = new NoneGrammar();
          } else {
            grammar = new NoneGrammar();
          }
        }
        diag("recognition and play started " + grammar);
        PlayAndRecResults pr_result =
          grammar.recognize(session,
                    noinput_timeout,
                    toomuch_timeout,
                    endspeech_timeout);
        info("recognition done " + toString(pr_result));
        switch (pr_result.rec_result.status.value()) {
        case RecognitionStatus._RS_SuccessfulSpeech:
        case RecognitionStatus._RS_SuccessfulDTMF:
          RecResultImpl result = new RecResultImpl(pr_result.rec_result);
          result.setActiveGrammar(grammar.getActiveGrammar(result));
          diag("recognition result=" + result);
          return result;
          /*
           // if any
           result.performAction(ip);

           value = result.getValue(field.getSlotName());
           if (value == null) {
           throw new EventException("event.nomatch");
           }
           break;
           */
        case RecognitionStatus._RS_NoSpeech:
          diag("error: no speech");
          throw new EventException("noinput");

        case RecognitionStatus._RS_TooMuchSpeech:
          diag("error: too match");
          throw new EventException("toomuch");

        case RecognitionStatus._RS_CallDropped:
          diag("call dropped in recognition");
          throw new EventException("telephone.disconnect.hangup",
                    "call dropped during recogition");
        case RecognitionStatus._RS_Error:
          error("internal error in recognition");
          throw new EventException("error.internalerror",
```

```
                        "fatal error in recognition");
            default:
              error("unknown event in recognition");
              throw new EventException("error.internalerror",
                        "unknown state in recognition");
            }
          } catch(SessionEnded ex) {
            throw new EventException("telephone.disconnect.hangup", ex.getMessage());
          } catch(MediaError ex) {
            error("error occured in playAndRecognition: " + ex.getMessage());
            if (ex.error_code == 3) {
              throw new EventException("error.semantic.noprompt", ex.getMessage());
            } else {
              ex.printStackTrace();
              throw new EventException("error.mts.media", ex.getMessage());
            }
          } catch(SpeechError ex) {
            error("error occured in playAndRecognition: " + ex.getMessage());
            ex.printStackTrace();
            throw new EventException("error.mts.corba.user", ex.getMessage());
          } catch(UserException ex) {
            error("error occured in playAndRecognition: " + ex.getMessage());
            ex.printStackTrace();
            throw new EventException("error.mts.corba.user", ex.getMessage());
          } catch(SystemException ex) {
            error("system error occured in playAndRecognition: " + ex.getMessage());
            ex.printStackTrace();
            throw new EventException("error.mts.corba.system", ex.getMessage());
          }
        }
    /*
        Slots toSlots(com.genmagic.invui.vxml.idl.invui.Slot[] s) {
          Slots slots = new Slots();
          if (s != null) {
            copy(slots, s);
          }
          return slots;
        }
        private void copy(Slots slots, Slot[] s) {
          for (int i =0; i<s.length; i++) {
            switch(s[i]._discriminator().value()) {
            case SlotType._T_String:
              sslot_s ss = s[i].sslot();
              slots.setSlot(ss.confidence,
                        ss.name, ss.value);
              break;
            case SlotType._T_Int:
              islot_s is = s[i].islot();
              slots.setSlot(is.confidence,
                        is.name, String.valueOf(is.value));
              break;
            case SlotType._T_Structure:
              rslot_s rs = s[i].rslot();
              copy(slots, rs.child_slot);
              break;
            }
```

```
        }
      }
    */
      private String toString(PlayAndRecResults r) {
        RecognitionResults rr = r.rec_result;
        StringBuffer buffer = new StringBuffer();
        buffer.append("status=" + REC_STATUS_STRING[rr.status.value()]);
        buffer.append(", num results=" + rr.results.length);
        if (rr.results.length>0) {
          buffer.append("{confidence=" + rr.results[0].confidence);
          buffer.append(", result=" + rr.results[0].result);
          if (rr.results[0].nl_interpretations.length>0) {
            buffer.append(", slots[0][0]=");
            append(rr.results[0].nl_interpretations[0].slots, buffer);
          }
          buffer.append("}");
        }
        return buffer.toString();
      }
      private void append(Slot s[], StringBuffer buffer) {
        for (int i=0; i<s.length; i++) {
          switch (s[i]._discriminator().value()) {
          case SlotType._T_String:
            sslot_s ss = s[i].sslot();
            buffer.append("<"+ ss.name + "," + ss.value + "> ");
            break;
          case SlotType._T_Int:
            islot_s is = s[i].islot();
            buffer.append("<"+ is.name + "," + is.value + "> ");
            break;
          case SlotType._T_Structure:
            rslot_s rs = s[i].rslot();
            buffer.append("struct " + rs.name + " {");
            append(rs.child_slot, buffer);
            buffer.append("}");
          }
        }
      }
      /**
       * Records user's input and return a VoiceXML variable associated with
       * recorded file.
       */
      public RecordVar record(int bos_timeout,
                              int tms_timeout,
                              int eos_timeout) throws VXMLException {
        try {
          check();
          diag("recording");
          long l = System.currentTimeMillis();
          ResourceManager rm = getAppContext().getResourceManager();
          ResourceManager.TmpFile tmp = rm.createTmpFile("tmp", ".wav");
          System.out.println("file = " + tmp.external);
          System.out.println("bos = " + bos_timeout + ", tms = " + tms_timeout+
                             ", eos = " + eos_timeout);
          RecordResult result = session.record(tmp.external,
                                               bos_timeout,
```

```
                                    tms_timeout,
                                    eos_timeout);
            info("recording done " + result);
            if (result.reason == RecordingDoneReason.RDR_NoSpeech) {
               throw new EventException("noinput");
            }
            if (tmp.external.equals(result.filename) == false) {
               tmp.internal = null;
            }
            return new RecordVar(result.filename,
                        tmp.internal,
                        (int) (System.currentTimeMillis() - l));
         } catch(SessionEnded ex) {
            throw new EventException("telephone.disconnect.hangup", ex.getMessage());
         } catch(UserException ex) {
            error("error occured in record: " + ex.getMessage());
            ex.printStackTrace();
            throw new EventException("error.mts.corba.user", ex.getMessage());
         } catch(SystemException ex) {
            error("system error occured in record: " + ex.getMessage());
            ex.printStackTrace();
            throw new EventException("error.mts.corba.system", ex.getMessage());
         }
      }
   /*
      synchronized private void waitForTransferCallConnection(long timeout) throws VXMLException, Timeout {
         long timeLeft = timeout;
         long until = timeLeft + System.currentTimeMillis();
         while (dropped == false &&
               transferState == CALLING &&
               timeLeft > 0) {
            try {
               wait(timeLeft);
            } catch (InterruptedException ex) { }
            timeLeft = until - System.currentTimeMillis();
         }
         if (transferState != TR_CONNECTED) {
            try {
               info("cancelling transfer call");
               trans_call.session.terminate();
            } catch (Exception ex) { }
            check();
            // throw new Timeout();
         } else {
            check();
         }
      }
   */
   /*
      synchronized private void waitForTransferCallTermination(int maxtime) throws VXMLException, Timeout {
         if (maxtime == 0) {
            maxtime = Integer.MAX_VALUE;
         }
         long timeLeft = maxtime;
         long until = timeLeft + System.currentTimeMillis();
         while (dropped == false &&
```